CPU power management for SMT and CMP processors

A study thesis by Dominik Winkelmeyer
supervised by Frank Bellosa and Andreas Merkel

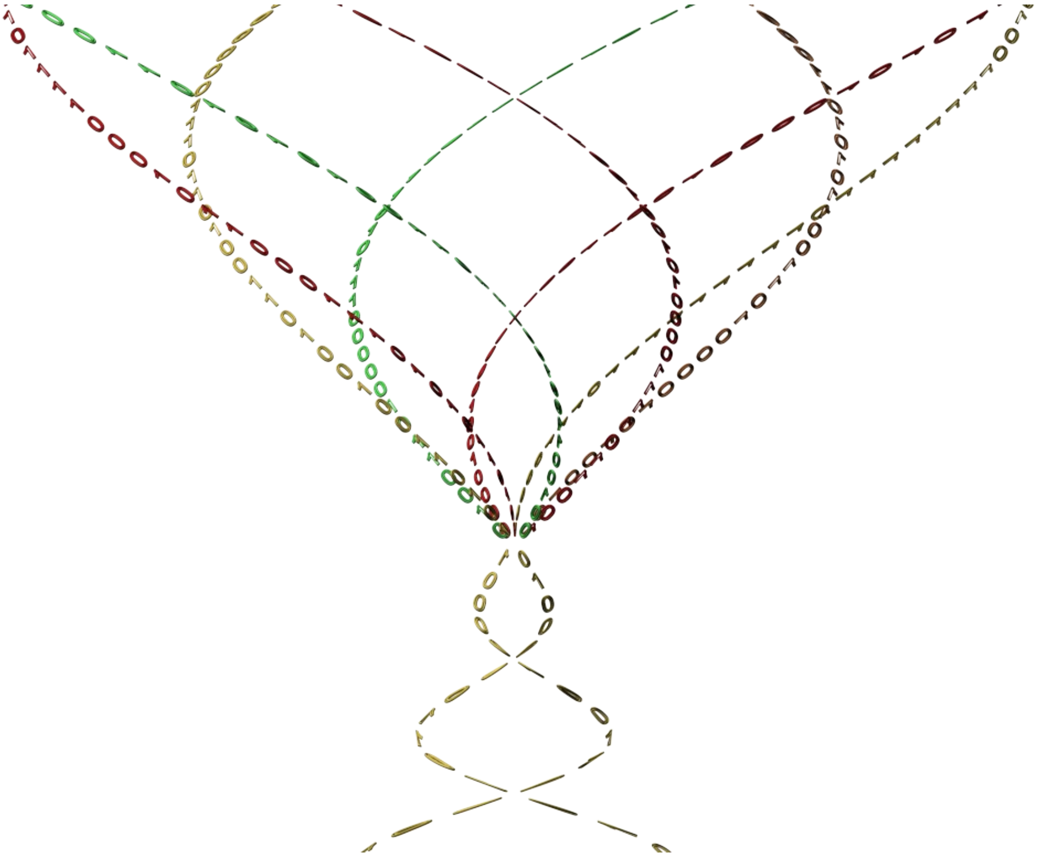

Table of contents

0 Abstract

This study thesis focuses on the power consumption of different combinations of applications on simultaneously multithreaded and multicore microprocessors. Therefore, performance monitoring counters are examined for their usability to estimate and predict the performance and power on SMT and CMP systems. A wide range of different processor events has been investigated to find some, that can be used to gather information about the behavior of the two architectures, when tasks share a microprocessor. These events are be used for an energy aware scheduler to decide, which tasks should be executed at the same time. Such tuples of tasks can be chosen to prevent peaks in the energy consumption or to reduce the ratio of power per instruction.

1 Introduction

The performance of microprocessors can be improved by increasing the processors frequency at smaller component size, but this increases the power dissipation per area dramatically and requires expensive cooling devices. This is one cause for todays trend to equip even private personal computer with more than one single processor core instead of increasing the processor's frequency or to improve the performance by a better utilization of components. In this study thesis, the energetic aspects of tasks running on such systems are investigated.

Performance monitoring counters are used in this work to analyze single applications as well as combinations of them. The application performance signature gained from this measurement is used to estimate the energy consumption of a single task or the whole microprocessor in real time. This approach is also topic of some related papers (see chapter 2). The goal here is to examine the possibility of predicting the energy and performance behavior of more than one application running on a microprocessor at the same time, based on the analysis of each single application. An operation system's scheduler should be able to use this information to limit the consumed energy by proper combinations of tasks, which are executed simultaneously on a SMT or CMP architecture. Such a strategy can stabilize the energy consumption of a microprocessor within a close range. This improves the ability to limit the power more precisely by using clock modulation or frequency scaling. Another aspect is the energy spent for each micro operation, which is especially on SMT processors tightly coupled with the kind of tasks that are executed simultaneously.

The setup of the test environment is described throughout chapter 3. To include side effects of running applications on each other in the measurement and its interpretation, a bottom up approach is used: many of the available events are counted to form a quantitative weighted model of the microprocessor's and the application's behavior. Chapter 4 and 5 contain information about how it was achieved to measure a large number of processor events and how the gathered data was interpreted and reduced afterwards. After that, the results from power and performance measurement are presented (chapter 6) and their connection to processor events (chapter 7). The consequences for energy aware scheduling on SMT and CMP architectures in chapter 8. It shows how performance monitor counters can help a scheduler decide which tasks should run at the same time on such systems.

2 Related work

This study thesis bases on the approach to relate consumed energy to processor events. Such events can be counted by many of todays microprocessors to analyze the performance of applications.[EDEA] describes the possibility to use some of these countable events for an estimation of energy a whole processor or a single task consumes in real time. This accounting can be used to limit the power or the heat dissipation using techniques like clock modulation or by inserting HLT cycles to the code a processor executes. Also the migration of tasks within distributed or multi processor systems can help to hold thermal limits for each component ([BPMP]). This work focuses on situations, where there are more tasks ready to run than a SMT or CMP system can execute at the same time. A scheduler can then choose which of them should be executed at the same time to avoid power peaks or reduce the energy a task consumes throughout its lifetime.

3 Test and measure environment

3.1 Hardware

There were two computer systems with two different microprocessors available for this work. As SMT processor, an Intel Pentium 4 (Prescot) with Hyper-Threading enabled was used, whereas an Intel Pentium D was used to represent a chip multi processor (CMP). The hardware layout of these two processors is very similar and they use the same mechanism for performance monitor counting, which makes a good comparison of these two architectures possible. Other microprocessors also offer event counting facilities, but the available events as well as the interface of accessing them differs for many microprocessors and manufacturers. There are approaches for machine independent event measurement like [RABB], but the processor events this work focuses on can most likely be found on other architectures as well.

The power of the processors was measured by an external system, which was set up to sample the energy consumption at 5 kHz. The values measured this way are the references for the correlations between performance monitor events and electric energy.

3.2 Usage of performance monitor counters

The Linux kernel 2.6.17.7 without forced preemption was used for the test computers. It was modified to support the access to performance monitor counters of the Pentium 4 and Xeon processors in user space. There were some additional requirements for this mechanism. First of all, the processor's events must be counted per task. Therefore the task_struct was extended to hold the configuration and the results of the performance monitor counters. The scheduler sets up the events for the next task at task switch after saving the results for the previous task. These two actions will be explained a bit more detailed now.

The bottom up approach of this work as described in the introduction, requires to count many different events. The Pentium 4 and Xeon processors support a maximum of 18 events to be counted simultaneously, but not every combination of them is possible. To use more events per application, they are round robin scheduled per task on every task switch. It is possible, that some events are not counted, because they occur at a point of time, where the schedule monitors other events. To avoid, that this influences the results, an endless loop layout is used for the test applications as described in the next chapter. When transferring the conclusions of this study thesis to real world applications (see chapter 7 and 8), there are not that many events necessary, so they do not need this special kind of layout.

To relate the events to the time for which they were counted, a time value is also saved in the task_struct for every type of event. This is based on the time stamp counter event of the microprocessor, which is read at task switch when saving the performance monitor counters. For the interpretation of the time measured this way, it is important to know how the time stamp counter works on the target micro architecture. On some processors this counter increments linear to time, on others linear to the clock rate of the processor, which may change. A cause for this change could be user- or kernel space power management software, or even a mechanism within the processor itself like Thermal Monitor 1 or 2 that throttles the processor, when a critical temperature is detected. On both of the test computers, the time stamp counter increments linear to time, which is also the planned mode of operation for future processors from Intel [IA32].

The performance monitor counters can also help to detect power state switches. On Pentium 4 / Xeon processors there is an event called power_events, that causes a counter to increment at every clock cycle. So the quantity of this event can be used to detect changes in the processor's clock rate. This is only possible if the time stamp counter increments linear to time. On some older architectures (like P6) it also increments linear to the processor's clock, but not on the SMT and CMP systems used for this work. The overall power consumption is not the main scope of this study thesis. Since here the events had been measured per task and there was no power management software running, many of these things were not relevant. Anyway, to assure there were no power state changes while one application from the test suite was executed, the power_events had been included in the test runs once per event schedule slice for each task. To set up the counters, a new entry in the proc filesystem was created, which can be used to configure a system wide default configuration. This is used for every new task but can be overwritten per task by another entry in the task's proc folder. It is also the place to read the counted events of a single task and the time for which they were monitored. To assure the consistence of the measured events, a mutex was used to lock the data while they are accessed by the scheduler or from user space.

3.3 Test suite

Eight applications are used to analyze the energetic behavior of the SMT and CMP systems: int, float, fileread, memcp, memwr, random, randomloop and randomcall. All of them have got a similar layout. After initialization each application enters an endless loop in which it performs the job the specific test was designed for. At the end of each loop a counter within the application is incremented. When receiving a SIGPWR signal, the application prints this loop count together with its clock time. After that, the loop count is reset and the clock time is saved to be subtracted from clock() on next SIGPWR. This way the applications in the test suite can give performance hints for every interval between two SIGPWR signals.

The suite can be divided into three groups. The first group contains two applications which perform integer and floating point operations respectively. The do not use much memory for variables, no I/O and no conditional branches within the main loop. The second group consists of three applications that mainly do I/O operations (from the processor's point of view). The first one reads a file from the hard disk to main memory over and over. The second application copies data from one to another location in main memory utilizing the standard library's memcp() call. The last member of this group is memwrite. It writes to main memory in a way that is very ineffective for the benefits of the cache hierarchy. The third group of applications, the random group, is used to analyze the efficiency of the branch prediction unit and the impacts mispredicted branches have got to other applications running on the same chip.

SIMD instructions like MMX, SSE or their extensions are left out of this work. The cause for that is, that on one hand their limits are expected to be dominated by the bus and main memory system. On the other hand there might also be situations, where the SIMD facilities within the processor might limit the speed of processing. One possible scenario for the later one could be this one: two tasks, running on a SMT processor make heavy usage of these instruction on a limited amount of data. Since the data could be cached, the SIMD components of the processor could give the limit for these two tasks. To investigate scenarios like this would go far beyond the scope of this study thesis so the SIMD aspects of the processors had been ignored, although they could be very interesting for aspects of consumed energy.

4 Measurement procedure

4.1 A note on processor events

The Pentium 4 / Xeon processors offer a facility to count events that occur while the processor is working. Some of the model specific registers are used to select and configure the events and the way they are counted. Other MSR's are used to count the events and can be read. The performance monitor counters of the used micro processor can count events when the path they are on is no more speculative. That means some events can be filtered for such, that occurred because of a mispredicted branch, or events that have been executed predicted. It is called “at retirement counting” and will be referred by this work in some places. Some events are called to be bogus in this context, when they occurred as a result of a branch misprediction. Events that arise on a predicted path are called to be non bogus. The SMT processor is able to filter some events by the thread that triggered them. These events are called “thread specific” while there are other events that can not be assigned to one of the threads. Such a type of events are called “thread independent” and can not directly be assigned to one specific task, if the threads of the SMT processor are executing different tasks. For a more in deep description of performance monitor counters see [IA32].

With the modifications to the Linux kernel described in chapter 3, it is possible to set up and read processor events per task via the proc file system. They are coupled with a time value based on the processor's time stamp event. In this study thesis, when referring to the quantity of a specific event, there is always the ratio of one event's occurrences per time meant. Since the time stamp event increments at the speed of the processors frequency, these values are in the interval of 0 to about 2 because some events can occur more than one time per clock cycle. The superscalar design of the used processors can as, an example, finish more than one micro operation per clock cycle. It is important to know, that the quantities of events can not be compared for different processors directly, because even when both increment the time stamp at a constant rate, they could use different factors of linearity for that. Anyway, they give hints to compare different microprocessors when keeping that in mind and since the design of the two processors used in this work is very similar, they also showed to be.

4.2 The test runs

Each application from the test suite was first executed in a single instance on each of the test computers. During this runs, an extern data acquisition system was used to read the power consumption of the processor. It was made sure, that no other tasks ran at the same time, which used any significant time on the processor. Because of the loop layout of the applications from the test suite, it was possible to read very constant values for the processor's power consumption. After that, these runs were repeated for every combination of two application from the suite running on the processor at the same time. Using Linux's CPU affinity feature, they were bound to different logical or physical cores. In this first test phase, also the performance of the applications was measured using their loop counters and clock values. The ratio of loop count to clock time will be referred in this text as an application's performance hint. To ensure its accuracy and reproducibility, each test ran for about one minute, after which it showed to be stable in most the cases (one exception, see chapter 6). The values of these performance hints are not directly comparable for two applications, since they spend significantly different spans of time for one loop. But they can be used to compare the performance of an application when executed alone or in combination with others.

Processor events were investigated in a second test phase, the motivation for this approach will be explained now. As mentioned before, the idea was to create a model of the processor, weighted by a broad bandwidth of measured processor events. It should rate the importance of single events for criteria, that can not be measured directly like the influences of processor threads on each other. This huge number of events should also show the most significant relations between single events and energy or the effects two tasks have got on each other. Chapter 3.2 described how these events are scheduled for the performance monitor counter registers. As the layout of the applications from the test suite is based on endless loops, the quantity of events (see chapter 4.1) should stabilize when running the test long enough. This is similar to the method used for the application's performance hints. The difference to this performance hints is, that there are some events, which occur very rarely and not at a constant rate. But these events should not be ignored in this test phase so a broadband application performance signature could be created. Anyway, running the tests long enough could solve this problem, but how long is long enough for such kind of sporadic events? To find out, a monitoring tool was created, which read the quantity of events from the proc file system each t seconds while the tests were executed. It could be configured to save the last n results of all event's quantities and keep on monitoring, while all events in this n results are within a range of less than i percent difference. The values used for the second test phase were in most cases t = 5, n = 10 and i = 1. This means, that the monitor sampled the event's quantities every five seconds, compares them with the last ten results and continued until the maximum value within this samples was smaller or equal to the minimum value multiplied by 1,01. It showed, that the spans of time for which the tests had to run to reach the required accuracy was much longer than for the performance hints and that there also were high variances of this spans, depending on the tested applications or their combinations. This monitoring tool, was the cause to measure the processor's power consumption and the performance monitor counters in two different test runs. To avoid the influence of the monitoring to the energy, the first run was made without it. But it should have had an effect small as possible to the performance monitor counters as well. Therefore the monitor was bound to the same processor as the application to be monitored. Since the events were saved on task switch, it was made sure, that not even thread independent events (for the SMT processor) produced by the monitor were accounted for the application of interest, because they never ran at the same time on the microchip.

Anyway more task switches to another application means more stress for each of them as cache lines must be refilled for example. Therefore some of the tests were replayed without the monitor, but for the same span of time they ran before. The results showed, that there indeed were slight differences in the measured quantities of events. On the one hand, they were in the range of 1 percent or significant less, depending of the test application and the event. So the effect could be measured especially in cache hit and miss rates but far not that significant for events such as the quantity of front side bus activities. These side effects of the monitoring tool seemed to be acceptable, especially since the same parameters were used for all tests, and therefore all applications had a comparable penalty.

This study thesis focuses on the energetic effects which could be achieved by scheduling of tasks. It would probably be of interest to compare these to the magnitude of impact of hardware based methods. Therefore some tests have also been repeated at different processor clock modulations.

5 Data analysis and interpretation

5.1 Number of events

For a real world scheduler it would be a bad idea to watch too many events. One cause for that is that a very complex model would use much computation to be evaluated, which might not be very useful for a frequently invoked part in an operating system like the scheduler. Another aspect is, that there are not only applications around, which are based on an endless loop layout like the applications from the test suite. So the scheduling of tasks should be based on results from fine granular and real time updated records to handle changes of an application's behavior. Using a large number of processor events conflicts with this requirement, because the frequency each event is measured declines with the number of events that have to be scheduled. Anyway, scheduling of processor events might not be necessary, if there are just few events to monitor. There are some scenarios for which an offline gathered application's performance signature could be useful as well. On example is to create a schedule on a batch system, to plan the execution of different but relative homogeneous tasks. Then it could be help to have an idea what these different tasks mainly do to decide, which can be executed at the same time for an effective use of energy and system resources.

5.2 Classification

For this study thesis it was required to reduce the number of events. The large number of them in the test runs was necessary to find out which ones are the best candidates for a prediction of the behavior of two applications with respect to consumed power. Excluding events does not necessarily mean to ignore the effect which triggered them. To heed this circumstance when preferring some of them, a classification is needed that should reflect the physical layout of a processor and a semantic arrangement of available events. An overview of the classification used in this work will be presented now.

- The number of µops (micro operations) executed by the microprocessor: these can be be divided into several sub groups. With at-retirement-counting, bogus operations can be distinguished from non bogus ones. Another point of interest might be which units of the processor are especially involved for their execution like the floating point unit or units for SIMD processing.
- The cache behavior of an application: in particular which levels of the processor's cache hierarchy are involved, what are their hit and miss rates and how do translation look aside buffers perform.
- The processor's I/O: how much traffic is there on the bus (or buses in case there are more then one) as a result of cache misses, prefetches or other activities like DMA operations.
- The application's predictability by the processor: especially how effective are the prediction units of the processor like the branch prediction. What is the best or worst case scenario for branch prediction and at which range are the ratios of predictions to mispredictions.

This list can give a general idea which kind of events were measured in the test phase of this work. Although many of the mentioned aspects are one to one covered by specific events provided by the Pentium 4 / Xeon architecture, some must be derived from others. The sum of all this aspects can give a good performance overview (which performance monitor counters have initially been established for), but only some events should be used to gather information of a processor's energy (see chapter 5.1) plus some to predict these for the combination of applications.

One class of events from the list above was not relevant for this study thesis: SIMD instructions. As described in chapter 3.3, there also was no application in the test suite to cover this aspect.

5.3 Bottlenecks

The speed at which an application is executed on a microprocessor is limited by different factors. The uppermost limit is given by the processor's clock speed and its pipeline layout. This means a processor can execute more than one instruction per clock cycle depending on the granularity of the pipeline, its functionality and the code to be executed. In many cases, this upper limit is not reached because, as an example, operations depend on data from the main memory, which has to be delivered via the bus. Some of these data might be available from one level of the processor's intern cache hierarchy duo to prefetching mechanisms or because they are already present from earlier accesses to nearby addresses within the system's memory that are loaded to the same cache line. In cases where an access to the bus has to be made, its speed or the one of the addressed resource limits the execution within the processor.

The speed of a task's execution is bound to one of these limits at all points of time. The kind of limit might (and in many cases will) change while the task runs. The idea for this work's goal to investigate the possibility of predicting the energetic behavior of tasks running together is to find events which show significant limits for each of them. They should then be used to make an assumption how the applications will influence each other and how this influences the processor's energy consumption.

There are two possible approaches to find the actual values for the limits that can be represented by processor events. The first one is to gather them from the specifications of a microprocessor and other system components. This seems to be a good idea in the first place, but in many cases it is very hard to achieve. If for example the number of clock cycles is known in which a second level cache line can be filled as well as the number of cache lines that can be filled simultaneously, then this information can give a maximum throughput of the cache, which will most likely never be reached by real world applications. The first paragraph of this chapter showed other scenarios, which makes this strategy very unhandy. In this work, benchmarking is used to estimate limits represented by processor events, since the applications from the test suite are nothing else than benchmark tools for special aspects.

5.4 Procedure of data reduction

In the first test phase (chapter 4.2) the power of the processors and the applications performances were measured for all possible combinations from the test suite, including tests for each application running alone. These tests have been repeated at different clock rates. The result was a good base of data for the energy and performance behavior of the test suite. Each single run in the second test phase took significant more time to achieve the required accuracy. Additional, in this second phase not only two values were measured per test but up to forty events. To reduce the runs in this phase, an analysis of the energy and performance data was made. It showed the points of interest and gave reference values for energy predictions. So in the second test phase, not all combinations were necessary to investigate. The quantities of events were measured only for each application running on a processor as a single instance and as two instances bound to different cores or processor threads. This made sense not only based on the data from the first phase, but also from a semantic point of view. The effects should be maximized by flooding the processor with code of one kind. Possible events indicating bottlenecks should show up using this procedure too. In addition to that, some hand picked tests were made where needed.

To analyze the data from the test runs, two aspects had been considered at all stages. One of them was the semantic of specific events and the knowledge of general functions and mechanisms within a microprocessor and of SMT and CMP architectures. The other aspect was the regard of events quantities. This was important for the approach to use a weighted model for the investigated processors as mentioned at the beginning of this text. It also showed to be very useful for a better understanding of some of the events. Although there are plenty of information available regarding the general setup and usage of performance monitor counters, the processor events are not that well documented. Based on these two aspects, it was searched for significant relations within the events quantities and between them and consumed energy (see chapter 7).

6 Power and performance

6.1 Motivation

Energy awareness can mean different things, dependent on the scale at which consumed power is considered. For an operating system's scheduler there are two reasonable focuses about energy: energy per time and energy per instruction. This chapter presents power and performance measurement results from the first test run which give hints about this general aspect for the SMT and CMP systems..

The data within this chapter refer to the highest energy level on the SMT (3 GHz) and the CMP (2,8 GHz) processor. Other tests showed, that the effects which can be observed here, become smaller at lower clock speeds. The diagrams presented in this chapter show the power or the performance off all applications from the test suite. They have all got a similar layout and contain data for all possible combinations of two applications from the test suite.

6.2 Power

The SMT and the CMP architectures show a very different energetic behavior when two tasks are running at the same time. While the power of the CMP system is very symmetric and predictable [Diagram 1], the SMT processor tends to have a more specific behavior when tasks are coupled [Diagram 2]. The interference of applications is more significant here, since the two processor threads share more resources.

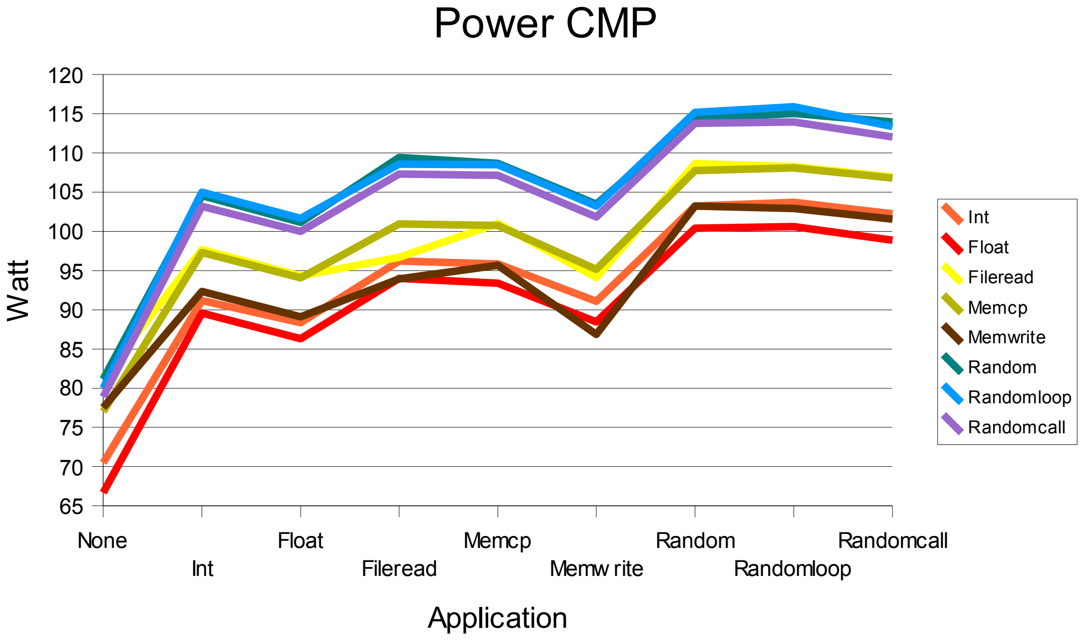

Diagram 1: Power of task tuples CMP

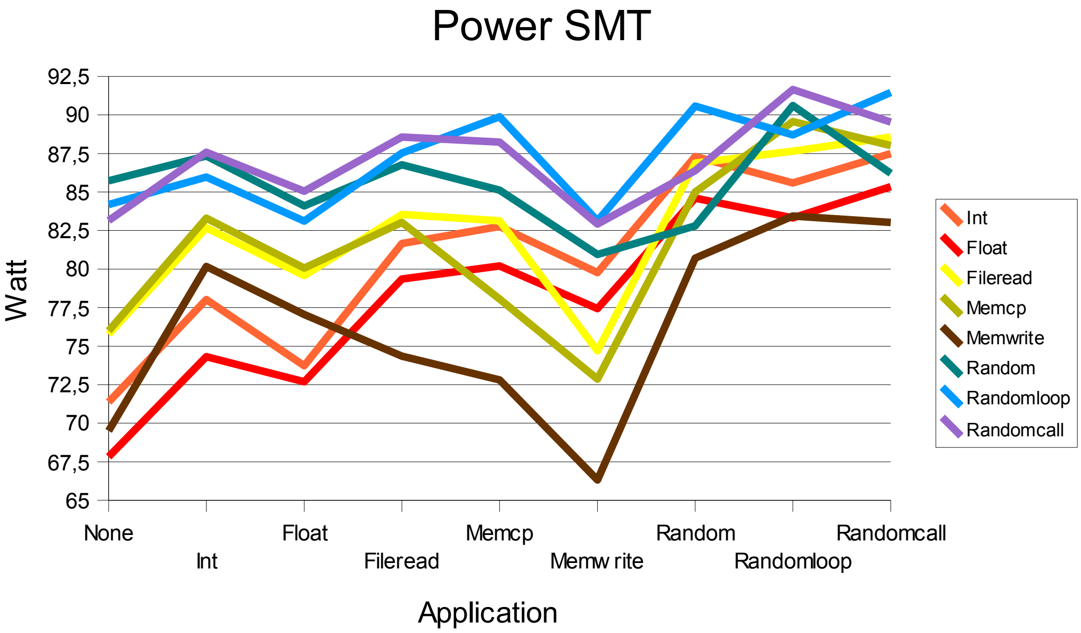

Diagram 2: Power of task tuples SMT

A model for predicting the energy consumption of two tasks will be described throughout chapter 7. One tuple of tasks is of special interest at this point: the combination of two instances of memwrite. On the CMP system, this tuple consumes just few energy compared to other tuples of two tasks, but on the SMT processor its power was even the lowest, including all single instance runs.

6.3 Performance

The next diagram [Diagram 3] displays the performance of the applications, based on their performance hints. The Y-Axis measures the speedup factor of two instances compared to the performance of a single instance for each application. These performance hints are calculated based on the loop count and the clock time of an application. Another diagram for the performance behavior of the CMP architecture, is not presented in this chapter, because all values are close to 1 (since the cores to not influence each other like processor threads). When handling the ratio of loop count to clock time on a SMT processor, there is one important thing to concern. If only one instance of an application is running, it can cover all the processors resources for the time it is executed. When two instances or two different applications are executed simultaneously, they have to share the processor's units within their clock time. This means that a speedup factor of 0,5 represents the case when performance equals a processor without SMT enabled.

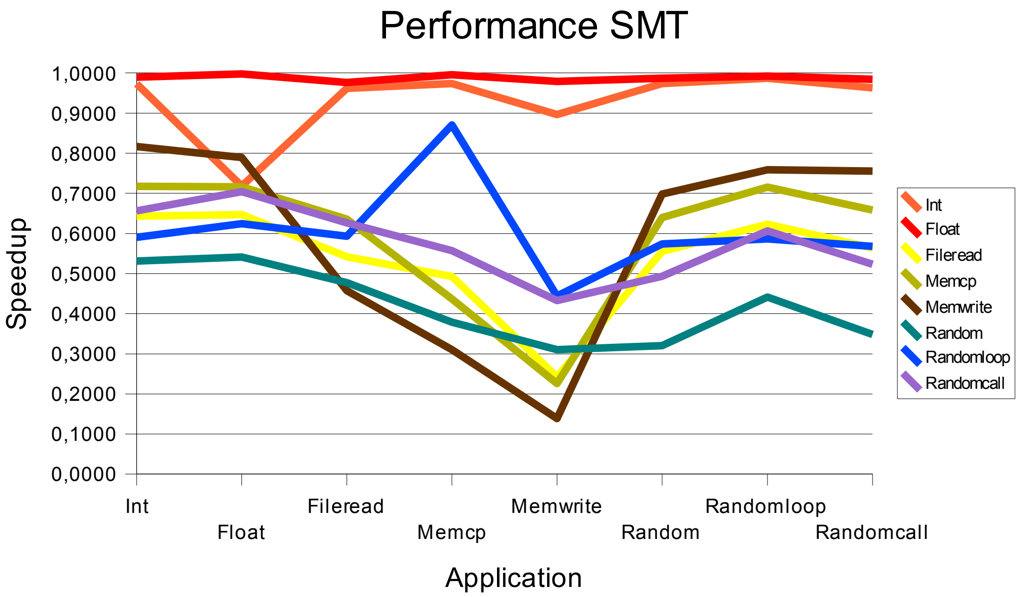

Diagram 3: Performance of task tuples SMT

Applications from the I/O group are the most vulnerable ones for heavy losses in performance when they are coupled with others. Two instances of memwrite have got a speedup of around 0,15 compared to one instance. It explains the measured power of this tuple (chapter 6.2): the two instances handicap each other so heavily, that the overall energy consumption of the processor is even below that of a single instance. While most applications show a performance loss when coupled with memewrite, especially the tests from the I/O group are slowed down significantly. This discourages the coupling of such tasks, since the losses in performance can hardly compensate the little amount of energy saved per time. The next chapter will present some more details about this aspect based on the analysis of processor events.

7 Processor events

7.1 Motivation and overview

Performance monitor counters can help a scheduler to decide which tasks should be coupled on SMT and CMP systems. Three of the aspects, which may be useful for such decisions will be covered in this chapter: the performance, the consumed energy and I/O operations. Performance is an important factor for energy awareness on SMT systems, when the coupling of tasks may lead to major drawbacks in performance. I/O operations and caching behavior on this architecture are important indicators for the side effects, that tasks could have on each other. The front side bus is an important shared resource on CMP systems as well, so there should exist knowledge about the magnitude of the interferences between active tasks on such an architecture. To couple tasks with respect to power, the probably most important factor is the energy consumption of single tasks. There already exist some work about task based power estimation using the performance monitor counters ([EDEA]). But based on the results from the second test run (where processor events have been investigated), a different model for energy accounting was used here. For this work, there are two benefits that come from this new approach: it shows the relation between energy and performance on the one hand, and less processor events are involved on the other hand. This chapter points out the processor events, that show a good correlation with these three aspects in the broadband analysis.

7.2 Hints about performance

The Pentium 4 / Xeon processors support an at-retirement event called uops_retired [IA32]. It counts how many micro operations had been retired within a clock cycle. Since it is an at-retirement event, the associated performance monitor counter can be configured to count bogus, non bogus or both kinds of events. When counting only the non bogus events, the number of micro operations are counted, that have been executed successfully. Bogus µops instead are operations that have been unnecessarily executed as a result of a mispredicted, so not taken branch. So the quantity of uops_retired is proportional to an application's performance, if only non bogus events are counted. [Diagram 4] shows this relation. It displays the value of performance hints when one single instance of an application is executed (solo), divided by the performance hints when two instances exist on the same SMT processor (duo). These ratios are named PerfHints, while there are also data for uops_retired, that refer to the ratio of non bogus micro operations when one or two instances run. It shows the correlation between non bogus uops_retired events and an application's performance. On the CMP system, all values on the Y-axis are close to 1, since the applications do not interfere that much with each other.

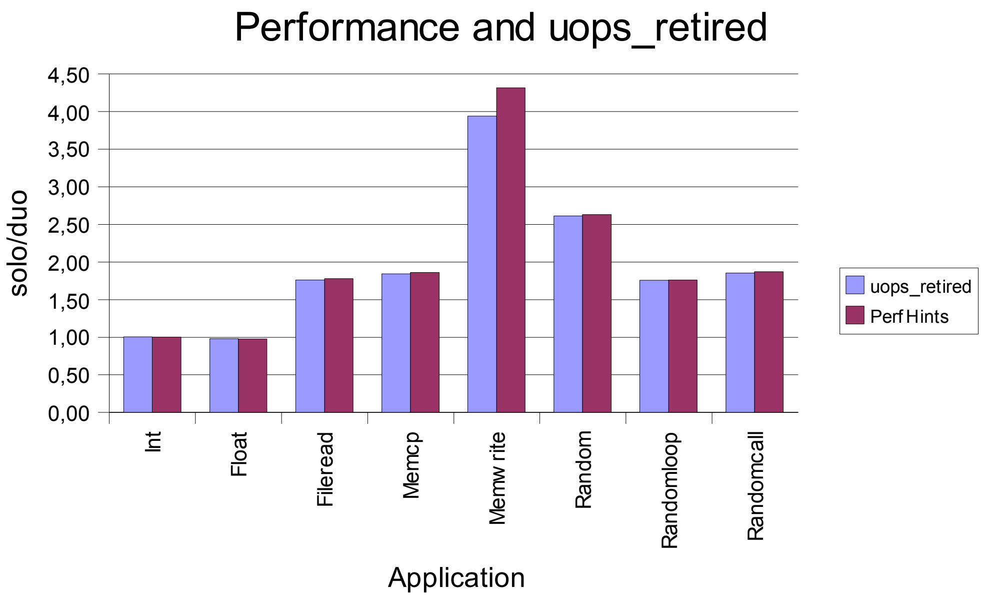

Diagram 4: Non bogus µops and performance

One word to the results of the memwrite application is necessary. This application's loop count increments significant less often than the one of every other application in the test suite. The effect was even much stronger when running two instances of memwrite. As you can see in the diagram: the performance of a single instance was about four times better, then when executed together with a second instance, which is based on the purpose it was designed for (see chapter 3.3). Therefore it shows a much bigger variance in the performance hints than all other applications. Longer runs could improve its accuracy, but since the performance was not the main concern of this study thesis, this case was (kept in mind but) accepted.

7.3 Counting Energy

The broadband counting of processor events showed the non bogus micro operations to be a very good indicator for an application's performance. We will now concern about the relation between consumed energy and processor events. The uops_retired events showed also to have got a strong relation to the processor's power. This makes sense, since more micro operations per time means more work to to for the microprocessor per time. But also other aspects will be discussed, which are useful to consider for the scheduling of tasks on SMT and CMP systems.

A processor consumes energy at a constant rate when it is executing no operations. This is the case, if it is halted for example or put to a deep sleep ACPI state. Linux 2.6.17.7, which was used for this work uses the MWAIT instruction within its idle thread if it is supported (which was the case for both systems). The MWAIT instruction can put the processor to an ACPI C state and use a monitor to return to normal operation mode. Within this state, a processor reduces its energy by one or more available strategies. The consumed energy of a processor equals about the multiplication product of its clock rate and the used voltage, so a processor can use techniques like frequency scaling or clock modulation to save power. At lower rates of speed, the transistors within a processor do not have to switch so fast anymore. It means, that they can be operated at a lower voltage level, which reduces the energy further.

These are important issues to consider when estimating the power of the whole processor using performance monitor counters. If there are only few operations executed within a specific amount of time, this could on the one hand mean, that the processor was waiting for external resources at a high power level, or on the other hand, that it executed this few operations and then entered a lower power level. From the operation system's point of view it should be no problem to distinguish the idle thread which utilizes a power save state from other threads (or tasks). But this does not cover all situations, in which the processor enters a low power state. Power management software could invoke a state switch or the processor itself to prevent damage due to overheating. The first case can be handled since the power management software should cooperate with the operating system, while in the latter case the thermal monitor can be configured to trigger an interrupt on state switch.

As mentioned above, the processor event which counts the retired micro operations can be used to monitor the processor's power. But there is a difference in the setup of uops_retired compared to the approach to measure an applications performance. There it was necessary to count only non bogus events, so micro operations which had not been executed based on a mispredicted branch decision. But all operations cause transistors to switches, including the ones on speculative paths. Therefore the uops_retired event must also be used to count bogus operations. There are two possible ways to achieve that. The first one is to set up one counter to count both, bogus and non bogus micro operations. The second approach is to use two counters, one for bogus and one for non bogus, which can be added. That way it is possible to collect information on the processors energy consumption as well as performance hints. But this second approach is not necessarily the better one for task specific measurement. The uops_retired event is a thread specific event on SMT systems. So it can be used to count micro operations either on one of two processor threads, or the other one or both. This makes nine possible setups from {t1, t2, both}x{non bogus, bogus, all} (t1 and t2 stand for each processor thread). But there are only two ESCR registers available on Pentium 4 / Xeon processors to count uops_retired events. This makes the first approach more attractive for task specific energy counting: for each of the processor threads, one register is set up to count bogous and non bogus events. For CMP systems this limitation is not necessarily given, since each core has got its own performance monitor counter's facilities. Anyway also each single core of a CMP system might be able to support SMT.

7.4 uops_retired vs energy, general results

Diagram [Diagram 5] shows the general relation between the processor's power and the uops_retired event. The data was gathered on the SMT processor with one busy thread, but the CMP system shows a similar behavior (see chapter 7.6). The X-axis shows the number of retired bogus and non bogus micro operations per time stamp event. The Y-axis shows the measured power of the processor in watt, as acquired by an external system. There are four graphs: the three upper ones are the linear trends of the test applications uops_retired events versus energy at different clock modulations. This means, that each application from the test suite was executed at a given processor speed and the quantity of retired micro operations was measured. The trend of these uops_retired events in relation to the consumed power makes one of these graphs. The actual results which based the trends are left out here due to clarity but will be presented throughout this chapter, when some aspects will be covered more detailed. At this point, diagram [Diagram 5] shall give an idea about the magnitude of the power ranges of the used micro processor with different clock modulations. The lowermost graph is a bit different. It is the nearly constant energy the processor consumes when the system is mostly idle (for 99% and above). So it does not depend on retired micro operations, since the MWAIT instruction as used by the Linux idle thread gets involved here. This graph was inserted to the diagram to conclude the general energetic overview.

The diagram shows the relation between micro operations and consumed energy, and there is a nearly linear relation between these two values. The trends are displayed for the ranges at which the tests produced results for uops_retired events. So it is understandable that the graph for 3 GHz reaches nearly the border of two micro operations per time stamp event, while the graph for 1,5 GHz does not cross the border of one micro operation per time stamp event. The gradient of the graphs is similar for different clock modulations of the processor, but there is another significant value for these energy levels, which will be referred in the following as the energy level offset. It is the point at which a trend would theoretically cross the Y-axis so where uops_retired equals zero (this is why the lines are extended to the left). The power ranges that are displayed in [Diagram 5] will give a good base to compare the effects an energy aware scheduler can achieve to the limitation of energy as a result of clock modulation.

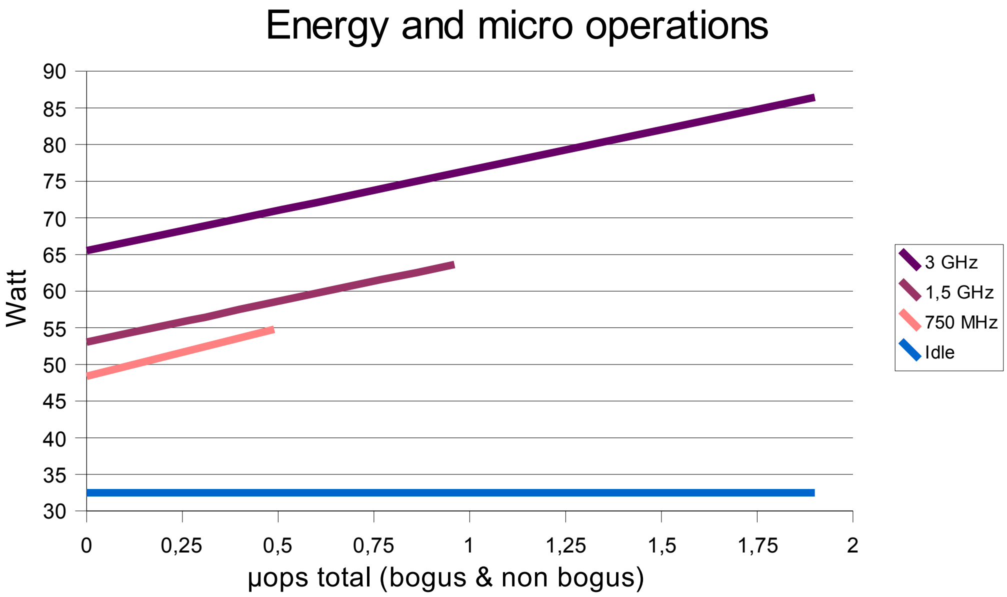

Diagram 5: Energy levels and µops general

7.5 SMT specific aspects

There are two results from the test runs, which are interesting for scheduling and energy estimation based on performance monitor counters on SMT processors. The first one concerns the relation between uops_retired and the processor's power. Diagram [Diagram 5] bases on data that were gathered in tests, where one application from the test suite was executed as a single instance, so the second thread of the SMT processor was idle. To show the relation of this two values when both thread are busy, another setup was used. Now each application was executed in two instances, each bound to another thread of the processor by using Linux's CPU affinity. Only the performance monitor counters on thread one were configured to count the uops_retired event for bogus and non bogus events for both threads. The monitoring tool was also bound to thread one, so the retired micro operations were only counted for this two instances of one application executed at the same time (the system had no other tasks which showed a significant activity). The test was then repeated with one single instance of each application, but still with uops_retired configured to count for both processor threads. For the case, another tasks runs on the second thread while this test, its micro operations should be counted too, since its energy consumption was also measured by the external acquisition system. This whole procedure was carried out at two different energy levels, at 3 GHz and at 1,5 GHz. The diagrams [Diagram 6 - Diagram 10] show the results of these tests. The keyword “solo” refers to results from tests where only one instance of an application was used, whereas “duo” marks results of two instances executed at the same time. The first four diagrams [Diagram 6 - Diagram 9] show the results of each single test run, containing the actual measured data (which are connected for clarity) and the resulting linear trend. This demonstrates the actual relations between the uops_retired event and the consumed energy of the processor, which were left out in chapter 7.4. The last diagram [Diagram 10] contains only the trends from the four tests.

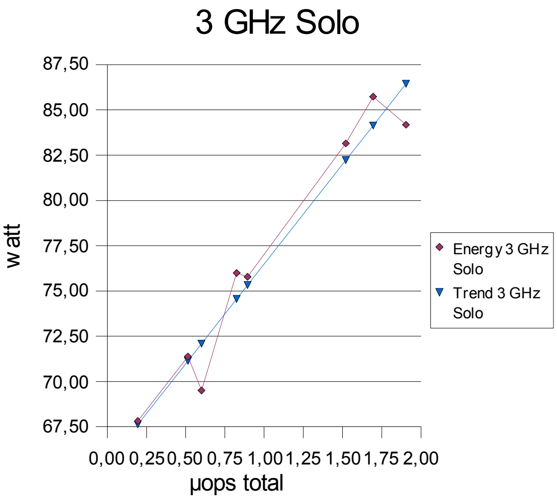

Diagram 6: Linear trend µops and energy

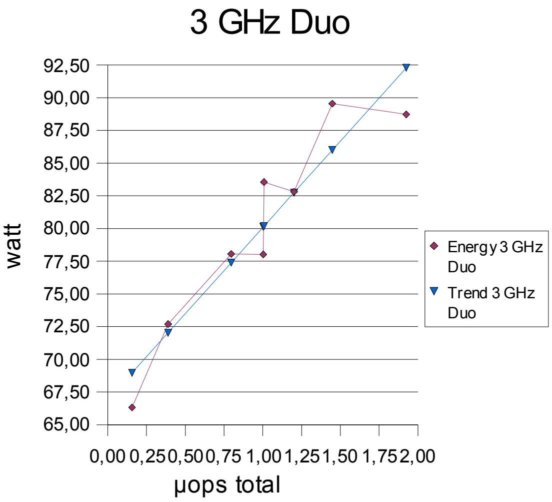

Diagram 7: Linear trend µops and energy

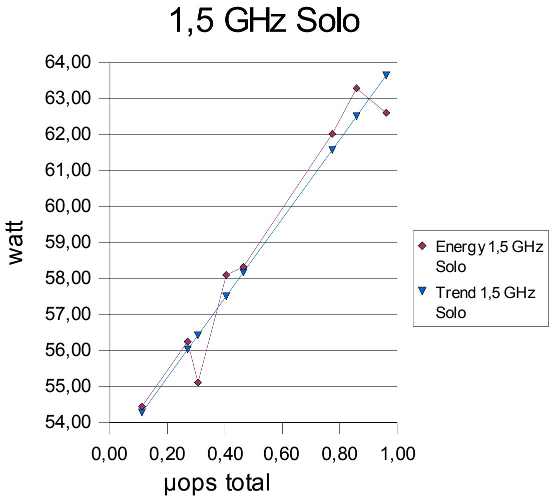

Diagram 8: Linear trend µops and energy

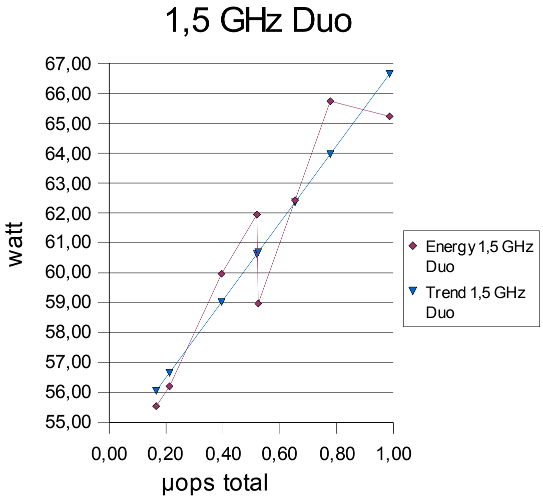

Diagram 9: Linear trend µops and energy

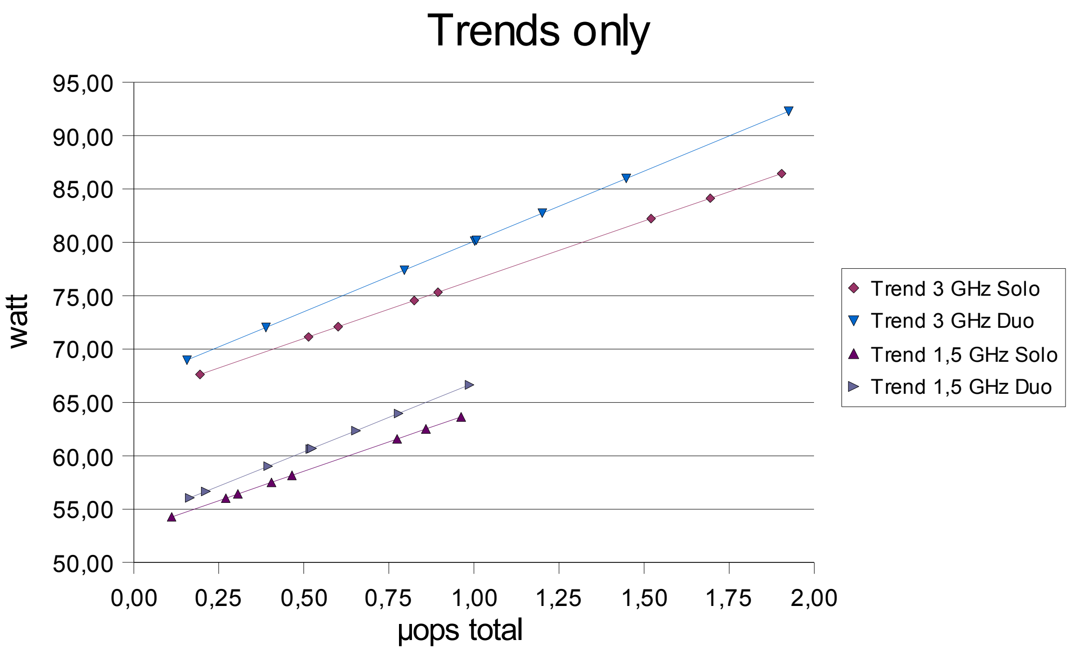

Diagram 10: Energy of SMT threads

The last diagram [Diagram 10] points out, that the relation between retired micro operations and the consumed energy differs on the SMT processor, depending on the load of the two threads. If both threads are used, each single micro operation should be associated to a higher energy then it would be if only one thread is busy. The trends differ from 2% at zero uops_retired up to 6,6% at two uops_retired per time stamp event at 3 GHz. For the lower energy level (1,5 GHz) this range is from 1,8% (zero upos_retired/ts) to 4,2% (one uops_retired/ts). This result should be considered for energy estimation based on the retired micro operations. To get a hint about the load on each of the processor's threads, the uops_retired event itself could be used, if it is counted thread specific.

There is another aspect on SMT processors about the relation between energy and performance. The processor switches between the two threads, so it is possible, that a result the first thread was waiting for will be available when the processor comes back to it. That way, the branch prediction might work better or the length of a mispredicted paths can be reduced compared to the case, where only one thread has got load. The following diagram [Diagram 11] shows the relation of four events measured at 3 GHz when an application is executed in a single instance compared to the case when it is executed in two instances at once (duo/solo). The micro operations (bogus and non bogus), the predicted and the mispredicted branches had been counted thread specific again, like in the most scenarios of this work, and only for one task in both cases. It is visible, that the ratio of bogus to non bogus operations is influenced by the load of the second thread as well as the ratio of predicted to mispredicted branches. Especially the I/O applications tend to drawbacks when they are coupled with other I/O tasks. If they are executed together with int, float or one of the random tasks, the prediction performs better.

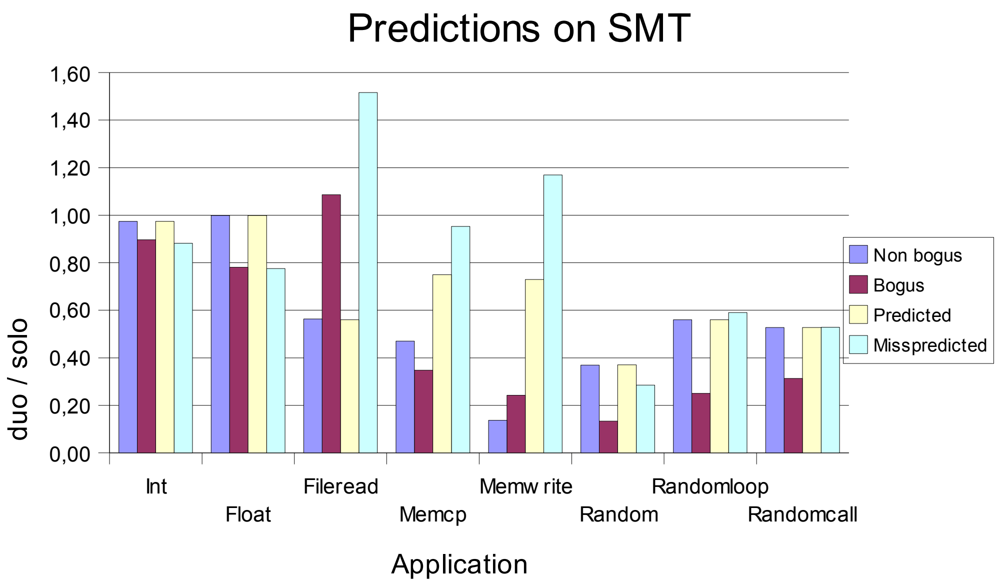

Diagram 11: Branch prediction effects SMT

7.6 CMP specific aspects

The CMP system shows a similar relation between the uops_retired event and measured values for energy. The setup for the tests on this processor differs just slightly from the setup used for the SMT processor. Since each core within the CMP architecture has got its own performance monitor counter registers, the events had been set up for each task in the case, where two instances of an application from the test suite were executed. The uops_retired events from both task have been added after the test, which explains that their quantity goes up to about four micro operations per clock cycle. There is one aspect, that is worth to be mentioned here. When only one core was used, there were three applications (marked in [Diagram 12], the point between memwrite and fileread represents memcp), which consumed more power than a linear relation between uops_retired and energy would predict. These three are the applications from the I/O class of the test suite. If the trend would only be based on the other five applications (trends in [Diagram 12] are based on all eight applications), the deviation of these three correlates very well with their quantity of the FSB_data_activity event. This effect however did not show up anymore, when the second core was busy to.

Like for the SMT processor (chapter 7.4), there should be a some words about the energy level offsets. The power consumption of the chip multi processor is about 43 watt when both cores are idle, while the energy level offsets are 70 watt for one idle core and 83 watt for both cores being busy. So from an energetic point of view, the processor uses a couple of shared resources for the cores. This encourages to use both cores for good energy per instruction ratios.

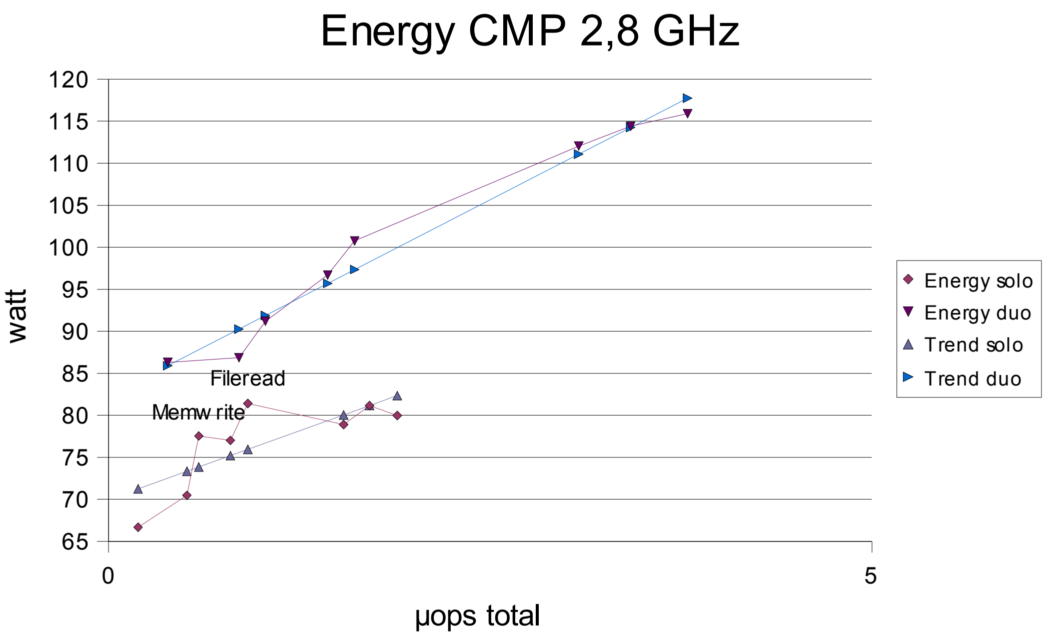

Diagram 12: µops and energy CMP

7.7 Accuracy of energy estimation

The processor event uops_retired was the one with the most significant single correlation to the consumed energy from the broadband test runs. This chapter pointed out some general results about this as well as some aspects, that have to be considered when using this event for estimation of the processors energetic behavior. The measurements also showed, that this correlation is nearly proportional [Diagram 6 - Diagram 9, Diagram 12]. Therefore the trend of the measured data was used to approximate straight line functions, which could be used correlate the consumed energy to the uops_retired event. The achieved accuracy of this approach is shown in the following table [Table 1] mathematically rounded to magnitudes of 1%. It contains the percentages of how far the measured results differ from this trends in the maximum and the average for the SMT and CMP processor as well as the medians of these distances. The data used for this table is the same as used in the rest of this chapter, so at every energy level and processor there had been eight tests for single instances and for two instances. The errors for the SMT processor are based on the corresponding trends for the thread's load as described in chapter 7.5.

The uops_retired event can be used for fine granular energy estimations, since it occurs at a very high quantity. Further improvements of the accuracy might be achieved by correction factors based on other processor events. The different levels of the cache hierarchy, accesses to the front side bus or mispredicted branches (and their correction) could be used for that. Also the operations themselves can be classified as load, store, floating point or SIMD operations. For this work, the achieved accuracy for energy estimation is presumed to be good enough, for two reasons: first of all, there are other papers like [EDEA], which cover such aspects much more precisely. The second reason is due to the focus of this work on the combination of tasks and its impact to consumed energy. Most of the limited performance monitor counter registers should be available to support scheduling decisions. With a scheduling mechanism of processor events like used in this work, the number of measured events is not a problem, but with every new slice for this event schedule, the granularity of the measured events gets worse. Although the accuracy of energy estimation based on processor events could be improved, this simple model will be used in this study thesis since it shows clear relations of energy, ACPI P states and performance. Counting bogus and non bogus micro operations with an single register can also give hints about performance, but there are situations, where this is very inaccurate (the portion of bogus micro operations is about 16% for randomloop, 11% for randomcall and 1,2% for random but all other applications show a rate of less than 1%).

Errors	Max	Min	Average	Median
SMT 3GHz solo	4%	0%	2%	2%
SMT 3GHZ duo	4%	0%	3%	3%
SMT 1,5GHz solo	2%	0%	1%	1%
SMT 1,5GHZ duo	3%	0%	2%	2%
CMP 2,8GHz solo	7%	0%	4%	4%
CMP 2,8GHZ duo	4%	0%	2%	1%

Table 1: Accuracy of energy estimation

7.8 Prediction, FSB, cache

The prediction of energy consumption for a specific tuple of tasks showed to be very useful for the CMP system, based on the model presented throughout this chapter. If an energy estimation of each single task was made, the resulting electric power of both tasks running simultaneously can be calculated for an energy level. This is done by calculating the linear trend function for an specific energy level and reading its value for the expected quantity of micro operations. But the front side bus is one shared resource of the cores, that can lead to an under- or overestimation. One aspect is, that the activity of the FSB unit consumes energy too and information about this might be used for an improvement of energy estimation for single tasks. When the coupling of tasks is considered, it is necessary to have some hints about the expected behavior of tasks within this new tuple. They might perform better or worse within this new tuple compared to the situation the energy estimation is based on, which means they will consume more or less energy. To include the side effects of tasks on the performance of each other in energy estimation and prediction, the resources of the system can be observed, that are or will be shared between them. An operation system could track the usage of system components like data stores or other peripheral devices. The performance monitoring within the processor can not be used to distinguish between such devices, but they can be used to observe the overall activity of the front side bus. This helps to classify tasks as more or less dependent of this shared resource, which helps to improve the predictability of power for a tuple. If the energy of one task with much FSB accesses was estimated when it was running together with another strongly

I/O depending task, it will most probably perform better when executed with a task, that uses the bus less. But when the task performs better (more micro operations per time), it will also consume more energy and vice versa, which should be regarded for power prediction. The Pentium 4 / Xeon processors provide a range of processor events related to I/O operations. One processor event that showed to be most suitable for such a hint about performance effects based on the coupling of tasks is called FSB_data_activity. It counts the number of clock cycles the front side bus is used, driven either by the processor or another resource (which includes another core on CMP systems). These two kinds of events can not be counted by one single model specific register on this architecture, so two performance monitor counters must be set up to include both kinds of accesses.

Interferences between tasks are more significant on SMT processors since more resources are shared by the processor's threads. Especially the common usage of the caches showed to be an important factor for this. This makes the predictability of energy less reliable, since even one available cache line fewer can lead to cache trashing. At the same time, this resource is very valuable for the ratio of energy per line of code, which makes it important for energy awareness. There are also some processor events, that can help to classify tasks by their cache usage. On the used systems, the second level cache and the third level cache (which was not actually available) can be watched for hits or misses, while the first level cache unit only provides information about misses. Hits on the first level might be derived from information of the other caches, combined with the tagging and filtering of micro operations for load and store operations (see [IA32]). A rating of tasks for their cache signature can be made based on such information. If one task shows only few hits and misses at one cache level, it will potentially leave more cache lines to another task executed at the same time (depending on the associativity, location of tasks in memory et cetera). Many hits or misses on the other hand do not necessarily mean, that many lines are used but improves the chance for that. There are other events and configurations available on Pentium 4 and Xeon processors, that allow to gather more detailed information about the cache usage, but such an analysis would go far beyond the scope of this work. Therefore, chapter 8 will present a classification based on hits and misses, that leads to good results for the coupling of tasks from the test suite.

8 Strategies for energy aware scheduling

8.1 Chapter overview

This chapter concludes the results of the measurement and outlines the consequences for a scheduling mechanism. It uses a separate examination of SMT and CMP relevant aspects. As shown in chapters 6 and 7, these two architectures differ significantly in the ability of controlling the consumed energy by a popper combination of tasks. The consequences for energy aware scheduling will be discussed in chapter 8.2, followed by a description of the strategies for the combination of tasks on each of the two architectures. These presented strategies are primarily based on results from the tests of the processor events, but focus more on scheduling compared to chapter 7. Events measured in the second testing phase have therefore been compared as indicators for the coupling of tasks as well as different methods to use these indicators. The tuples of tasks from the test suite that would result by these strategies have then been rated by the energy consumption and performance measurement from the first testing phase (see chapter 6). The most promising strategies for the SMT and the CMP systems are described in chapters 8.3 and 8.5. Furthermore there is a discussion about how a scheduler can account thread independent events to single tasks, which might be necessarily on SMT (or hybrid) processors.

8.2 Power per time or power per instruction

The results from chapter 6.2 show the potential to reduce or limit the processor's energy consumption with a scheduling strategy. But when when considering this solution, the question is, if this approach makes sense or if it brings to much penalties, especially for the SMT processor. The answer can be found in chapter 6.3 when correlating the energy saving with the loss in performance. A limitation of energy can only be achieved by the combination of applications which handicap each others performance very strongly.

When this approach is inverted and good combinations for performance are focused, there are much better results for energy per instruction then there are penalties for energy per time. Chapter 7 did already show this relations which had been approved there on a wider base of data. When having a look at [Diagram 5] again the following conclusion can be made: if you could speed down an application by 100% (from two uops_retired to zero) the maximum rate of energy that could been saved is at about 25%. This is the best possible result from the model of chapter 7, since there are just few applications which come close to the border of two micro operations per clock cycle. The cause for that is the high energy level offset. Now the conclusions for an energy aware scheduler of these results will be discussed. The SMT relevant aspects will be followed by the ones for CMP.

The general idea for energy aware scheduling on SMT processors as it is presented within this study thesis is to maximize the number of micro operations per clock cycle at the highest allowed or lowest necessary energy level of a processor. It means to use tuples of tasks, which have got the smallest influence possible on each other. This influence takes place in shared, slow and / or limited resources of the processor or the whole system. So the queue of ready tasks should be searched to find couples, that are expected to work fine together. This could be done by greedy strategies like first fit or by more complex ones like introducing other queues to achieve a best fit over all the available tasks for scheduling. There might even be situations, when it is better to execute just one single task on the processor. One example for such an scenario could be this: there is a task which reaches nearly the limit of possible micro operations per clock cycle. This task could show a high hit rate on the second level cache but only few misses. If all other ready tasks also show a significant use of the second level cache and also many misses but only few micro operations per cycle, they will most likely slow down this one task without much benefit for them self. Since this specific task already has got a good energy per instruction ratio it could be wise to give the whole processor to it for its time slice. This will probably raise the bogus operations but make each operation less power consuming (see chapter 7.5).

For CMP systems, another strategy as for SMT processors is possible. The difference to the latter ones is that there are less shared resources within the processor, which makes the decisions of the scheduler less complex and could focus more on energy limitations by proper combination of tasks. Anyway if there are just ready tasks to choose from which are expected to show comparable high energy consumptions, there are some actions possible to consider. Since the cores of a CMP could in most cases set to different energy levels independently, one or both of them could be throttled down, to match the expected energy consumption of two tasks. Another strategy could be to put one core completely to sleep and use just the other one, for example if there are few tasks and most of them are blocked most of the time. [BPMP] shows the benefits of such a strategy if there are more available cores than busy tasks. Although there was no processor investigated in this work which combines the SMT and CMP aspects, it is imaginable that the presented strategy could be useful for such ones, especially for cases to decide if it is better to use just one core or just one thread per core.

When the best tuples are found and the power consumption for each of these is estimated, the energy could be limited by the use of an adequate energy level. There might be the case in which only some tuples within the planned schedule will break this limit. In that case these tuples could be grouped and executed at a lower level than the rest. This depends on the length of the time slice and the time it takes to throttle the processor. Eventually these two groups could be executed in an alternating order on every schedule circle to minimize the number of energy level switches.

8.3 SMT scheduling

We will start here with the predictability of power and performance behavior. This has got two aspects: the estimation of energy for a specific energy level (ACPI P state) and the prediction of the influences of two tasks on each other. With the model from chapter 7 to relate processor events to energy, a good assumption can be made for the energy consumption on different energy levels of the processor. When for example one tuple of tasks have an uops_retired quantity of 1,5 at 3 GHz, it would have a quantity of 0,75 at 1,5 GHz. This means that it would consume about 63,5 Watt compared to 87 Watt when the clock is modulated down by a factor of 0,5. This makes it possible to choose a proper energy level so a tuple of tasks (or even one single one) holds a given energy limit. The predictability of the consumed energy for a specific combination of tasks showed to be not that accurate on the SMT processor (see [Diagram 2]). Two examples should help to outline this. When executed alone, int has an energy consumption of about 72 watt, memwrite 69,5 and memcp 76 watt. But when executed as tuples, int and memwrite consume 77,5 watt whereas memcp and memwrite consume only 73 watt. This could be explained since memwrite and memcp show a high usage of the front side bus and the cache, so they handicap each other much more than int does (which shows less activity in both cases). Now to the second example. The most power consuming applications are random (86 watt) and randomloop (84 watt). But the combination of randomloop and randomcall is significant higher than the combination of randomloop and random. Based on the wide range measurement of processor events, it was possible to find the cause for many of these effects, but they differ from tuple to tuple especially in their magnitude of impact.

On the way to find combinations with respect to performance (chapter 8.2), there have been two ideas. First of all, only tasks should be coupled which are expected to not interfere which each other by showing significant usage of resources which could be a bottleneck. Second, to maximize performance applications should be coupled that way, that there are many micro operations within each tuple. The most important bottleneck was of cause the cache. On the Pentium 4 / Xeon architecture there are some ways to get information about it. It is possible to count misses of the first, second and third level cache as well as the hits of the second and third level. The first level cache misses did not show to be that useful for information about the cache usage, so the second level was the most important unit to watch (the Pentium 4 used for this work did not have a third level cache). The hit rate on the second level gives good hints on the usage of the cache in most cases but the miss rate should also be watched, since also a cache miss fills up a cache line. With these two values, the applications from the test suite have been ordered by their cache usage. Then the application with the most significant cache signature could be coupled with the one with the least one. One word to the front side bus activities. This could be another bottleneck, but comprehensions with the CMP processor, where every thread has got its own cache showed, that it could nearly be neglected. The shared cache it the most fragile shared resource for performance on SMT processors. The front side bus events could also be seen as a kind of subclass of the cache usage: a task with a high front side bus activity will most likely have also a significant cache signature, but not necessarily vice versa. This must of cause not be true for tasks which invoke many DMA operations, which may block the bus without involving the processor's cache. Anyway, the cache seems to be the most weighty factor for the side effects of tasks on each other.

A similar approach can be considered when looking out for tuples, that should maximize the number of micro operations per time. The ready tasks can be ordered by the quantity of the uops_retired event and then the one from the top of the list could be coupled with the last one. Anyway, it showed, that there is another important event which should be used when sorting such a list. This event is called uop_queue_writes and counts the micro operations that are written to the queue of the processor. It appeared that two applications which have a low quantity of uops_retired, but a high quantity of uop_queue_writes do not perform that good as tuples, which show fewer queue writes. So the micro operation queue of the processor, that holds delayed operations (e.g. until all operands are available) could also be assumed to be a kind of bottleneck.

When bringing the two aspects mentioned together, there are some algorithms which are tested for the applications from the test suite. It was assumed that there is one instance of each application ready to run and each would use a full time slice, so there are no decisions for overlapping tuples necessary. This is of cause a simplified model, since especially the task with I/O operations like fileread could block immediately after they got the CPU. These effects should be considered for a real world scheduling algorithm which can be developed by extending the algorithms presented here. One of the most promising algorithms for performance (energy per instruction) will be explained now. It is a two level strategy, which first sorts the ready queue by second level cache hits. This list is then split up in the middle, so there is one part which contains the tasks with few and one with much cache activity. The cache miss rate could be ignored in this case, because it would not have had an effect on which task is in which part of the list. Anyway this first step is used to separate the tasks by their cache usage. So they could for example be sorted by the result of (second level hits + 50 * second level misses). The factor of about 50 should (based on the measurements) weigh the misses just slightly stronger than the hits and will result in the same resulting list parts for the scenario used here. Now, after the tasks are classified by their cache usage, the second step is to optimize the rate of micro operations. As described above, there are two relevant events which count the operations themselves and how many of them are enqueued. So each of the two parts of the list is now ordered by (uops_retired/uop_queue_writes) separately. The tuples to be used are now found by combining the first task from one part of the list with the last one from the other part of the list, the second with the penultimate and so on. This strategy does not result in the theoretically achievable performance optimum for the test suite but shows good results compared to other investigated algorithms. It is also considerable to sort the task in the second stage by the quantity of uops_retired, but at least for this scenario an index based on (uops_retired/uops_queue_writes) produced better results, also compared to strategies, that focus only on the cache usage. Strategies that combine tasks with a different quantity of retired micro operations will in most cases also prevent the worst case energy peaks (the presented strategy does it), but the limits of prediction on SMT systems does not make them very reliable for keeping the consumed energy strictly below a given value, as it is possible with ACPI states.

8.4 Thread independent events on SMT processors

When scheduling decisions are based on processor events, there might arise a problem on SMT processors with thread independent events. These events can not be filtered by the processor thread that causes them and therefore it is hard to assign them directly to a specific task. The problem is solved if just one task is running on the processor. This can occur for example, if there is only one task in the ready list. When there are more ready tasks available, one possible solution could be to give the whole processor to a single task once in a while in order to measure these thread independent events. Since the lost in performance for a SMT processor is not that big if one thread is unused as it would be on a CMP architecture, this might be possible. It might also not be necessary to do so for all the time, since there might always occur the situation where only one task is ready. Other tasks might not use one full time slice and the measurement could take place only for an fraction of one full slice, which reduces the penalty further. But this approach does not only reduce the performance but also the fine granularity of the measurement. In addition, the behavior of the task might change when it is executed with another afterwards. To include these effects, another strategy will be described in the following.

In chapter 8.3 was described which events are most important for scheduling decisions on SMT processors. These are events counting micro operations relevant data on the one hand, and on the other hand events for gathering information about the cache usage. The events for the micro operations are thread specific, so they do not cause any trouble. The cache relevant events however are thread independent, so they can not directly be assigned to a specific task. The presented strategy for selecting tuples however does only use the cache fingerprint to distinguish two classes of tasks and does no strict order by the cache usage in the end. So this classification could be done not on based on tasks but on tuples. If one tuple shows a heavy usage of the cache, both tasks could be assumed to be cache relevant in the first place and vice versa. They will then be coupled with tasks of lesser cache usage next time. For each task there could be a statistic, if the tuples they are assigned to do always produce much cache activity or not. That way a good assumption of the relative cache usage per task could be made over two or just a few time slices. But this statistics should not cover too many schedule cycles since the idea about the locality of software could not be extended to the full lifetime of a task. In a real world scheduler, there will of cause be more overlapping tuples than in the model used here. Therefore such a kind of rating task tuples could even be more efficient. Think of tree tasks A, B and C. When A gets the CPU, B is already running and this tuple shows much cache activities. The while A is still executed by on SMT thread, B is replaced by C on the other thread. This new tuple of A and C do just few cache hits and misses, so it should not be a bad idea to classify B as heavy cache user.

8.5 CMP scheduling

The scheduling strategy on a CMP architecture offers more possibilities and better results in limiting the consumed energy by the combination of tasks that are executed on the cores at the same time. The two most important causes for that are, that there are many exclusive resources for each core including performance monitor counter facilities and the possibility of using core independent ACPI states. Based on the model from chapter 7 to relate energy to retired micro operations there are good predictions possible, how much energy a specific task tuple would consume. There is one resource which is shared even on the CMP architecture, which is the front side bus. Three applications from the test suite were designed to measure the magnitude of impact this could have on the energetic behavior of applications but it showed, that an instance of memcp does not produce much front side bus activities, when it does not share its cache with another task like on the SMT processor. This leaves only fileread and memwrite, and therefore only tree possible combinations of their instances for measurement, which discourages a quantitative hypothesis about the impact of the front side bus. Anyway the available data show, that it might be possible to use the front side bus events of the processor to predict the magnitude this resource influences a specific task. The most promising event for this is FSB_data_activity. The performance monitor counter assigned to this event can be configured to count events related to data on the bus which will be sampled by the processor or which will not be sampled, however not both at the same time. This makes it eventually necessary to use two counters to get an idea about the traffic on the bus, but this would for example prevent the usage of global_power_events. From the operating system's point of view, this must not be a problem, since clock modulations driven by the processor may arise an interrupt and changes of the energy level from within the operating system (e.g. idle thread, energy aware scheduler) or the user space should be noticeable anyway.

Although the scheduling strategy for the CMP architecture as it is presented here focuses on the limitation of energy, the performance should not be totally ignored. So there are two arguments to couple tasks which differ in their usage of the front side bus like the classification of the cache signature from chapter 8.3: the first one is to achieve a good energy per instruction rate and the second is to keep the influences of each task on one other small to improve the quality of prediction. To hold a specific energy limit after the tasks are classified that way, there are some strategies possible. Tasks with the lowest expected energy consumption could be coupled as long as they will hold the limit. When there are just tasks left which will break the limit, the proper energy level (ACPI P state) for one or both cores could be calculated to ensure the limit. In the next schedule circle the procedure could take place the other way round to an increased energy level. Another approach is to use a more best fit oriented algorithm which tries to find tuples in a way, so the limit will not be broken without a state change of the processor. If this fails, the processor can be set to another energy level. These approaches depend on a fast responding state transition like clock modulation. If this is not available, or the timer frequency is very high, there could be another approach, which bases on the ability to set the ACPI states of the cores independently. They could be set to different energy levels and the most power consuming tasks could be assigned to the slower core. This implements a more fine granular limitation of energy than it would be possible with just one core, especially when combined with the assumed power consumption of each task. Although it prevents many state transitions compared to the first strategy, the two cores should change their role after some period of time to gain a better spreading of heat over the processor, which will result in a better efficiency of the cooling device and equal stress on the cores. The best strategy of the discussed ones depends not only on the overhead an energy level transition could eventually cost, but also on the timer frequency and the number of ready tasks running on a system. If there are only tasks ready, that show a huge variance in the durations until they become blocked, a first fit strategy which relies more on ACPI states to enforce energy limits would probably be the better choice than a best fit strategy which could produce more overhead. In situations where the tasks show a very homogeneous behavior over time, a best fit strategy, focusing on a proper coupling of tasks could do the trick. For the scenario of chapter 8.3 (one instance of each of the test applications is ready to run), a schedule can be found that holds a limit of at least 8% below the most power consuming task tuples all the time. The strategy to find this schedule was, to classify the tasks by their front side bus activity, sorting them by quantity of micro operations, and coupling them like in chapter 8.3 (the first with the last and so on).

9 A brief overview

This study thesis showed the benefits that can be achieved by energy aware scheduling on SMT and CMP systems, but it also described, that the mechanisms and goals that should be used for these two architectures are different. On a SMT processor the main target of coupling tasks is good performance, since this results in good energy per instruction ratios, while power limits should be enforced by ACPI states. For CMP systems, an energy aware scheduler can prevent energy peaks without a noticeable impact to performance. Together with the good predictability of power on such an architecture, this can be used to hold energy per time limits without involving the clock rate of the CPU.

10 Appendix

References

EDEA: Frank Bellosa, Simon Kellner, Martin Waitz, Andreas Weissel, Event-Driven Energy Accounting for Dynamic Thermal Management, 2003
BPMP: Andreas Merkel, Frank Bellosa, Balancing Power Consumption in Multiprocessor Systems, 2006
RABB: Don Heller, Rabbit - A Performance Counters Libraryfor Intel/AMD Processors and Linux, 2006, http://www.scl.ameslab.gov/Projects/Rabbit/
IA32: Intel Corporation, IA-32 Intel® ArchitectureSoftware Developer's Manual Volume 3B: System Programming Guide, Part 2, 2006

Diagrams