PERFORMANCE GAIN AND ENERGY SAVINGS USING MULTI-THREADED SOFTWARE ON AN EMBEDDED ASYMMETRIC CHIP MULTIPROCESSOR

Michael Bogner[(a)], Christof Steps[(b)], Franz Wiesinger[(c)]

[(a),(b),(c)]University of Applied Sciences Upper Austria – Embedded Systems Design
Softwarepark 11, A-4232 Hagenberg, Austria

[(a)]michael.bogner@fh-hagenberg.at, [(b)]christof.steps@fh-hagenberg.at, [(c)]franz.wiesinger@fh-hagenberg.at

ABSTRACT
More powerful computers, as well as the rising costs for energy require modern computing systems to be more and more energy efficient. A modern approach in order to solve this issue is the usage of multi-core processors instead of single-core processing units.
This paper focuses on modern ARM multi-core processors, which combine powerful and energy efficient processor cores on a single chip, in order to reduce power consumption and increase performance. To verify how much energy can be saved and how much performance can be gained by using multi-threaded software, we have simulated a various number of different calculation scenarios for a modern ARM System-on-Chip.
The results of these simulations show notably, how advantageous the usage of multi-core processors and power saving processors together on a single chip is. By using intelligent multi-threading concepts, the processor is able to reduce its power consumption by up to 60% compared to a single-threaded execution.

Keywords: multi-core, parallel software, power consumption, embedded asymmetric processor

1. INTRODUCTION

Modern computing systems, especially mobile devices, are getting more powerful every year, which results in an increase in power consumption. Furthermore, modern batteries still don't provide an acceptable power capacity and energy costs are constantly rising. Considering all these factors, processor manufactures tend to develop processors that are more energy efficient.
But being energy efficient alone is not the way to go, if the processors performance is lacking. A modern mobile device for example spends 80% of the time in an idle or standby mode, where only small background tasks are being executed. On the other hand, tasks that require a lot of performance should still be executed as quickly as possible. This trade-off between a low power consumption when doing light work but still providing a great performance when needed is the current aim for modern processor manufacturers.
One way to achieve this is to use multi-core processors. By using more than one processor core, power consumption can be significantly reduced by sharing work across all available cores. Those cores can run at a much lower clock speed and therefore use less power than a single-core processor, whilst still providing the same or even more performance.
Another way to be more energy efficient is to use both high performance and low power processor cores together on a single chip. With this architecture, the processor can save a lot of energy during low performance tasks, but still provide maximum power with its high performance processor cores. An example for this architecture is the NVIDIA Tegra 3 processor. It uses 4 high performance ARM Cortex-A15 processor cores and one low power ARM Cortex-A7 processor core on one chip. This setup is called 4-PLUS-1 (Nvidia Corporation 2011). The advantage of using these cores together is remarkable: During high performance tasks, the Cortex-A15 quad-core unit provides maximum power. While the processor is in idle mode or not doing any heavy work at all, its power consumption can be reduced dramatically by switching to the low power Cortex-A7 core. However, both types of processor cores cannot work together at the same time. This means that the processor never has more than four active cores, because either the Cortex-A15 quad-core, or the single Cortex-A7, so-called "companion core", is active. The operating system itself only sees 4 logical processor cores, without knowing if it's the single companion core or the four high-performance cores (Nvidia Corporation 2013).
The advantages of these setups are huge: By reducing the power consumption, mobile devices will have longer battery lives, less charging cycles and of course smaller batteries, which lead to a thinner and lighter device. Especially modern "internet of things" devices as wells as intelligent embedded systems will profit a lot from a reduced power consumption, because they are independent from constant power supply.

The objective of this study was to simulate different workloads for the NVIDIA Tegra 3 processor, in order to figure out the best way using the processor energy efficiently. This includes single- and multi-threaded

applications as well as simulating heavy workloads and measuring the processors performance, energy consumption and processor allocation. The results are then being compared and discussed.

2. RELATED WORK

Energy efficiency and performance especially on mobile devices have become increasingly important throughout the last years. As a result, many works were published throughout the last years, dealing with multi-threading to reduce power consumption (Youssef, A. et al. 2010, Grant et al. 2006, Arm Ltd. 2013). Those works however differ from this paper, as they are using simulators for the hardware, instead of real hardware and are taking a more theoretical approach. Focusing on using different processor cores on the same chip, there has also been done a lot of work in the past (Nvidia Corporation 2011, Nvidia Corporation 2013, Arm Ltd. 2013). Those works again have a more theoretical approach, as they are lacking tests with real hardware, which is the focus of this paper. Other tests using modern processors for mobile devices as well as real software prototypes have not been conducted yet.

3. TEST ENVIRONMENT

Power consumption and energy efficiency are very important in all modern computing systems, including mobile devices as well as desktop computers. As mobile devices suffer the most from a high power consumption, due to limited battery capacity, these tests focus on an NVDIA Tegra 3 processor. The Tegra 3 processor is, for example, part of an HTC One X Phone, which was used for these tests.
The used operating system is Android and the simulation software is developed with the Java programming language.

3.1. Clock Speed and Processor Utilization

In order to evaluate the results of the simulation tests, an application to measure the processors clock speed as well as its utilization had to be developed. Unfortunately, the operating system itself does only see four processor cores, even though there are five logical cores. In order to overcome this issue and get the usage and clock speed of all five cores, we took a closer look at the processor cores clock speed: while the high-performance Cortex-A15 operates at a clock speed between 1000 and 1500 MHz, the low-power companion core operates at a clock speed of around 350-750 MHz. By taking this fact into consideration, we can easily determine which processor cores are currently active. For example, if there is only one active processor core, and its clock speed is below 750 MHz, we can assume that we are currently operating on the low-power companion core. On the other hand, if there are more than one cores active, or all of the active cores clock speed is above 1000 MHz, we can assume that the high-performance quad-core is currently enabled.

The processors total utilization can be read from the Android system file "/proc/stat", as seen in Listing 1. The four values represent time units in which the processor was in a certain state since the device had started. The first value represents the time the processor spent in the user state, then system state, nice state and finally idle state.

Listing 1: Content of the system file /proc/stat

```
1 cpu 8000 2000 1000 9000
```

To calculate the relative time the processor was busy and idle, two measurements have to be made and the values have to be subtracted, as shown in Table 2. The Subtracted values now show the relative time the processor spent in a certain state over the last 0.5 seconds.

Table 1: Two measurements of the CPU utilization and the subtracted values over the last 0.5 seconds

	User	System	Nice	Idle
Measure 1	8000	2000	1000	9000
Measure 2	9500	2500	1500	9500
Subtracted	1500	500	500	500

This means that during the last 0.5 seconds the processor spent a total of 3000 time units either working or idle. Using this value and the time the processor spent in each state, the relative processor utilization can finally be calculated:

Idle: 500 / 3000 = 16 % (1)
Busy: 2500 / 3000 = 83 % (2)

To avoid influencing the systems performance by reading the processor utilization too often, the calculations are done every 0.5 seconds. By doing so, the values are still representative and don't influence the systems performance. Figure 1 shows the application on the home screen. The application always stays on top of every other application and shows the clock speed as well as the utilization in real time, updating every 0.5 seconds.

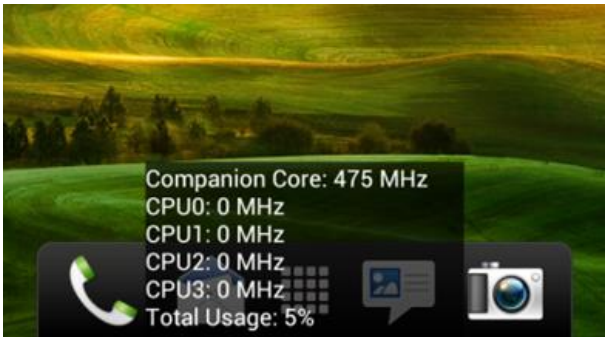

Figure 1: The measuring application on the homescreen.

3.2. Power Consumption

In order to evaluate the power consumption of the processor, an application called "PowerTutor" was used

(Lide Zhan et al. 2010). This application shows the power that certain parts of the device use, such as the processor itself, the Wi-Fi connection or the display. It calculates the processors power consumption approximately using the operating voltage and the current utilization. Figure 2 shows a screenshot of the application. The green chart represents the power consumption of the processor over the last 60 seconds.

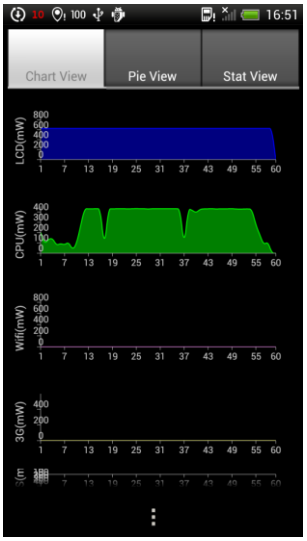

Figure 2: The PowerTutor application.

3.3. Objectives

The objective of this paper is to evaluate the processors multi-threading capabilities as well as its energy efficiency in different scenarios. Thus, we developed several tests to measure the processors multi-threading performance as well as its power consumption and efficiency. Firstly, we took a look at the performance that can be gained when using multi-threading, in order to find out if using more threads will result in an overall greater performance. Secondly, we compared the power consumption of the companion core and the high performance cores. This test aims at finding out if doing more work on the companion core, and therefore sacrificing performance for a lower power consumption, is advantageous in certain scenarios. Next, we analyzed the energy that can be saved when using multi-threading. Furthermore, we tested how the processors dynamic load balancing works. The goal of testing the processors dynamic load balancing, which means its efficiency, is to find the best way for an application to be executed: at high speed with maximum performance but a very high power consumption, with reduced speed but an overall lower power consumption, or a mix of both. The following sections evaluate these tests and discuss the achieved results.

4. PERFORMANCE GAIN THROUGH MULTI-THREADING

This test should show the performance gain when doing heavy work on multiple threads, compared to a single-threaded execution.

4.1. Simulation

In order to test the processors multi-core performance appropriately, an application was developed, that simulates heavy workload. This simulation includes prime number calculations and square root calculations, as those operations require a lot of performance.

The algorithm for the prime number calculation was realized using the well-known division method. In order to check if a certain number is a prime number, it is divided by every number up to the square root of the given number. Is the remainder of one of these divisions zero, the number is not a prime number. The algorithm itself is supposed to have a long execution time, in order to simulate a heavy workload and is therefore not very efficient. The square root algorithm simply involves multiple calls of Math.sqrt() on a range of numbers.

In order to get appropriate execution times, those calculations are being called on up to one million values. In the single-threaded test, one thread does the whole calculation. When testing the multi-core performance using more than one thread, the workload is split evenly among all threads.

The multi-core performance is measured using the speedup. The speedup is a mathematical formula that describes the relation between a serial and a parallel execution time of a program.

4.2. Results

The results of the multi-core performance tests are shown in Table 2. The execution times of the algorithms are given in seconds. As expected, the results show that the usage of more threads significantly reduces the execution time of the algorithms. Furthermore, using more than four threads does not speed up the calculation any more, even though the Tegra 3 processor has five logical processor cores. This is due to the fact that, as already mentioned, the Tegra 3 does not use all five processor cores at the same time (Nvidia Corporation 2013). The fifth thread, however, will then be running on one of the main cores instead, resulting in a lower performance as with four threads due to oversubscription.

Table 2: Multi-threaded execution times in seconds

Threads	Prime number	Square root
1	6.550	5.821
2	4.013	2.910
3	2.887	2.002
4	2.125	1.498
5	2.201	1.522

Figure 3 shows the achieved speedup. When using two threads, the program executes approximately 150% faster than on one thread. Three threads are more than 200% faster and four threads are around 300% faster than the single-threaded execution. Using five threads or more does not speed up the calculation any more.

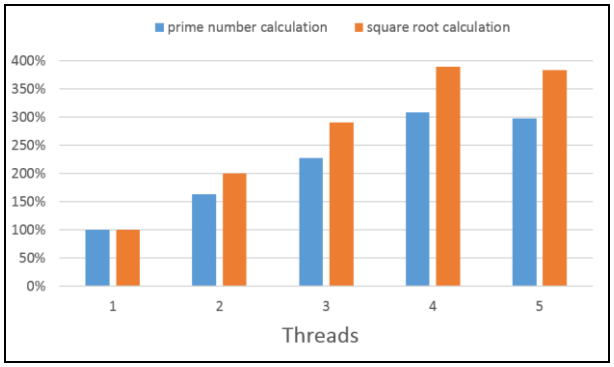

Figure 3: Graphical representation of the achieved speedup in percentage.

5. POWER CONSUMPTION: COMPANION CORE VS HIGH PERFORMANCE CORE

In this test we analyzed the power consumption of the processor in various different scenarios.

5.1. Simulation

The first scenario analyses a single-threaded application with a very low workload. The goal of this is to force the processor to use the companion core. As the operating system does not know anything about the five processor cores (and so the companion core), it can't decide which core to use, because the processor core allocation is done entirely in hardware. But by using a very simple low workload application, the hardware scheduler disables the high performance cores and enables the companion core.

The second scenario analyses a single-threaded application that does a heavy calculation. The goal here is to use one high performance core and measure its power consumption. For this calculation we used the prime number calculation already mentioned in section 4.1.

The last scenario analyses the same calculation as in the second scenario, but the workload is now split among four threads. The power consumption of the processor, as well as the processor core allocation is then being measured for 60 seconds.

5.2. Results

The results of the different scenarios are shown in Table 3. Just as intended, with the low workload, the hardware automatically executed the code on the companion core and therefore saving a lot of power. Over the course of 60 seconds, the processors average power consumption was around 50mW, which can be seen in Figure 4. The power consumption was staggering between 0 and 100mW.

Table 3: The processors power consumption in different scenarios.

Processor	Power Consumption
1x Companion Core @ 475Mhz	50 mW
1x HP Core @ 1500Mhz	380 mW
4x HP Cores @ 1400Mhz	400 mW

The single high performance core, however, needed around 380mW of power on average throughout the 60 seconds, which can be seen in Figure 5. The processors clock speed during the calculation went up to 1500 MHz, which is only possible in single-core mode. When two or more high performance cores are active, their clock speed is limited to 1400 MHz.

As we can see in Table 3, the multi-core power consumption, using all four cores, is around the same as the single-core power consumption. This means that, even though we could increase our performance by around 300% compared to the single-threaded execution, the power consumption still stays the same, no matter how many of the high performance cores are currently active. This is very important point for modern software development: By splitting work among multiple threads, the processor can reduce the clock speed of the processor cores and therefore save a lot of power and still provide the same or even more performance.

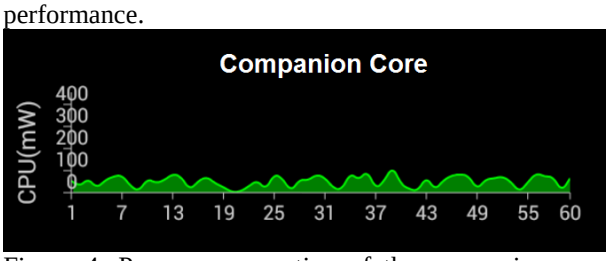

Figure 4: Power consumption of the companion core over 60 seconds.

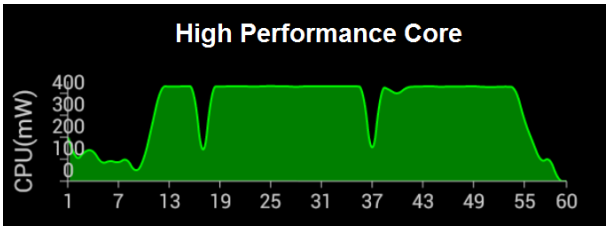

Figure 5: Power consumption of the high performance core over 60 seconds.

6. POWER SAVINGS THROUGH MULTI-THREADING

This test shows how much power can be saved by using a multi-threaded instead of a single-threaded execution.

6.1. Simulation

The simulation involved a prime number calculation up to one million values. The calculation was simulated on one, two, three and four threads, in order to measure the power consumption of the different multi-core scenarios. The workload was split evenly among all available threads. The result we are looking at is the average power consumption over the time that the single-threaded execution took. This means, if the calculation takes 60 seconds on a single-core, we take the average power consumption of these 60 seconds as a reference. Doing the calculation on two cores is faster and therefore takes less time to execute, but we are still taking the full 60 seconds into consideration for the power consumption. This means we are comparing all

scenarios over the full 60 seconds, even though the calculation might finish early in some scenarios.

6.2. Results

As already mentioned, a single-core computation at 1500 MHz has the same power consumption as a multi-core computation at 1400 MHz. Furthermore, we can save power when distributing the workload among multiple cores, as Figure 6 shows. The fact that the multi-core calculation is faster, enables the hardware to disable the high performance cores and enable the companion core earlier. The faster the execution is, the earlier the companion core can be enabled, which saves huge amounts of power, as the results in section 5.2 already showed.

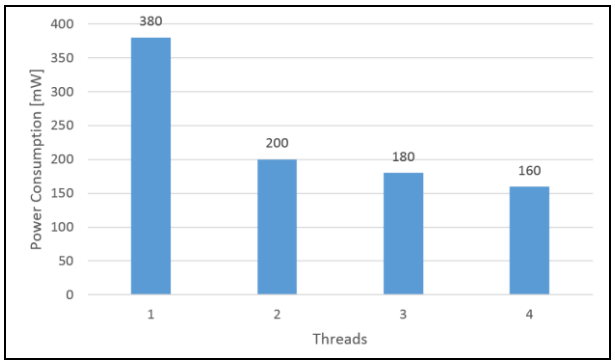

Figure 6: The average power consumption over the time of the execution, with the workload being distributed among all available threads.

Figure 7 shows how long certain processor cores where active during/after the calculation. The single-threaded execution took 60 seconds, so 100% of the time the Tegra 3 processor used the high performance core, which results in a fairly high power consumption.
When using two threads, the calculation only takes half the time and the processor can enable the companion core at around 30 seconds. When using four threads, the calculation finishes after 15 seconds and therefore the processor spends the following 45 seconds on the companion core, saving a lot of power.

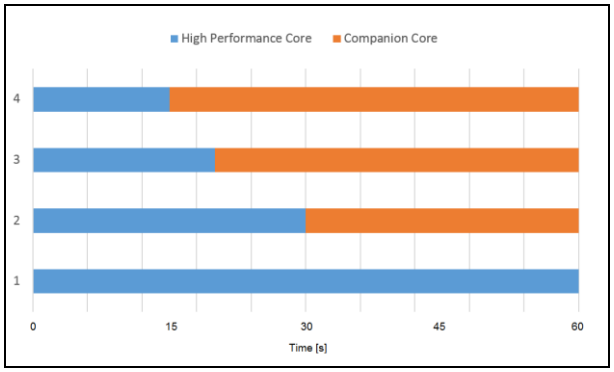

Figure 7: The time the device spent using the high performance cores and the companion core.

When looking at the results in Figure 6, we can clearly see how much power can be saved using more than one thread. When using two threads, the power consumption can be reduced by 50%, and with four threads it can be reduced by more than 60%.

7. DYNAMIC LOAD BALANCING

This test should show in how far the processor core allocation and the dynamic clock speed can be influenced by software development. We want to find out at which load the hardware scheduler switches from the companion core to the high performance cores, and when the clock speed is increased.

7.1. Simulation

The application for these tests does a square root calculation on a dynamic number of values. Additionally, to dynamically reduce and increase the load, short delays are added between every calculation. The application starts with calculating the square root of 1000 values, including 10 ms break after every calculation. The number of values then has been increased and the delay between the calculations has been decreased after every test, in order to simulate an increasing workload.

7.2. Results

The results of the dynamic load balancing tests can be seen in Table 4. With only 1000 square roots to calculate and 10 ms of breaks in between every calculation, the entire operation is done on the low power companion core. By increasing the number of values and reducing the delay, the hardware scheduler switches to the high performance core with a clock speed of 1 GHz. When further increasing the number of calculations, the processors clock frequency increases to 1.5 GHz with a load of 70%. With 100.000.000 square roots to calculate and no delay, the processors load goes up to 100%, using the maximum performance.

Table 4: Results of the dynamic load balancing test.

Values	Delay	Processor	CPU Load
1.000	10 ms	Companion Core @ 475 Mhz	18%
100.000	100 us	Companion Core @ 475 Mhz	40%
100.000	10 us	Main Core @ 1 GHz	50%
1.000.000	1 u s	Main Core @ 1.5 Ghz	70%
100.000.000	-	Main Core @ 1.5 Ghz	100%

This example shows very well, how the software can influence the processors core allocation as well as the clock speed. By adding short breaks in between calculations or reducing the number of calculations, we can force a complex operation onto the low power companion core, and therefore save a lot of power.

8. CONCLUSION

Based on the results of these tests, a lot of interesting conclusions regarding parallel software development can be made.

Generally, as far as the given algorithms allow it, multi-threading should always be preferred over a single-threaded execution. Using more than one thread on a multi-core processor increases the performance by a lot, and also greatly reduces the power consumption. As the results show, a single-core ARM Cortex-A15 at 1500 MHz clock speed uses the same amount of power as a quad core ARM Cortex-A15 at 1400 MHz. Therefore, it is always better to use multi-threading.

Another factor is dynamic load balancing. When calculation times are shorter, the processor itself can go into an idle state much earlier, and reduce the clock speed drastically or even switch to a low power core to save more energy.

When developing background tasks or tasks that should be executed when the device is in idle or in standby mode, reducing the workload can save a lot of power as well. This can be realized by adding short breaks in between calculations. For example, if you are doing some heavy calculations in a loop, the hardware scheduler will most likely use the high performance core for this operation. But by adding shorts delays after every loop, we can force the scheduler to stay on the companion core and therefore save power. This can be done if the calculation time is not very important or the result is needed at some time later in the future.

For applications that periodically read data or do calculations, increasing the interval can also reduce the CPU load and therefore force a lower clock speed or different processor cores to be used.

Finally, it can be said that using low power and high performance processor cores together on one chip is definitely the way to go. As the results of these tests show, the advantages that this setup brings to the table are huge, especially for mobile devices. But not only mobile devices profit from a setup like this, the results of these tests can be translated to desktop computers as well. Even though they do not use asymmetric multiprocessing at the moment, dynamic load balancing and multiprocessing in general have been part of desktop processors for years.

All in all, the processor saves a lot of power when not doing any heavy work, but still provides maximum performance when needed. But in order to profit from these technical advantages, the software has to be adapted as well. The software developer has to have knowledge of the underlying architecture in order to profit from all those advantages.

REFERENCES

Lide Zhan et al., 2010, Accurate Online Power Estimation and Automatic Battery Behaviour Based Power Model Generation for Smartphones. Available from: http://robertdick.org/publications/zhang10oct.pdf [accessed 4 May 2015]

Nvidia Corporation, 2013, NVIDIA Tegra 4 Family CPU Architecture. Available from: http://www.nvidia.com/docs/IO/116757/NVIDIA_Quad_a15_whitepaper_FINALv2.pdf [accessed 4 May 2015]

Nvidia Corporation, 2010, The Benefits of Multiple CPU Cores in Mobile Devices. Available from: http://www.nvidia.com/content/PDF/tegra_white_papers/Benefits-of-Multi-core-CPUs-in-Mobile-Devices_Ver1.2.pdf [accessed 4 May 2015]

Nvidia Corporation, 2011, Variable SMP (4-PLUS-1). Available from: http://www.nvidia.com/content/PDF/tegra_white_papers/Variable-SMP-A-Multi-Core-CPU-Architecture-for-Low-Power-and-High-Performance.pdf [accessed 4 May 2015]

Nvidia Corporation, Nvidia-Tegra-3-Mobilprozessoren. Available from: http://www.nvidia.de/object/tegra-3-de.html [accessed 4 May 2015]

Thomas Küneth, Android 4: Apps entwickeln mit dem Android SDK, 2012, Galileo Computing.

ARM Ltd. 2013, Multi-threading technology and the challenges of meeting performance and power consumption demands for mobile applications. Available from: http://www.arm.com/files/pdf/Multi-threading_Technology.pdf [accessed 04.05.2015]

ARM Ltd., ARM Processor Architecture. Available from: http://www.arm.com/products/processors/instruction-set-architectures/index.php [accessed 20 March 2014]

ARM Ltd., 2013, big.LITTLE Processing with ARM Cortex-A15 and Cortex-A7. Available from: http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf [accessed 20 March 2014]

Youssef, A. et al., 2010 M., On the Power Management of Simultaneous Multithreading Processors, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.18, no.8, pp.1243,1248, doi: 10.1109/TVLSI.2009.2020727

Grant, R.E.; Afsahi, A. 2006, "Power-performance efficiency of asymmetric multiprocessors for multi-threaded scientific applications," Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International , vol., no., pp.8 pp.,, 25-29 April 2006