QuickIA: Exploring Heterogeneous Architectures on Real Prototypes

Nagabhushan Chitlur, Ganapati Srinivasa, Scott Hahn, P K Gupta, Dheeraj Reddy, David Koufaty, Paul Brett, Abirami Prabhakaran, Li Zhao, Nelson Ijih, Suchit Subhaschandra, Sabina Grover, Xiaowei Jiang, Ravi Iyer

Intel Corporation; Contact: bhushan.chitlur@intel.com

ABSTRACT

Over the last decade, homogeneous multi-core processors emerged and became the de-facto approach for offering high parallelism, high performance and scalability for a wide range of platforms. We are now at an interesting juncture where several critical factors (smaller form factor devices, power challenges, need for specialization, etc) are guiding architects to consider heterogeneous chips and platforms for the next decade and beyond. Exploring heterogeneous architectures is challenging since it involves re-evaluating architecture options, OS implications and application development. In this paper, we describe these research challenges and then introduce a heterogeneous prototype platform called QuickIA that enables rapid exploration of heterogeneous architectures employing multiple generations of Intel processors for evaluating the implications of asymmetry and FPGAs to experiment with specialized processors or accelerators. We also show example case studies using the QuickIA research prototype to highlight its value in conducting heterogeneous architecture, OS and applications research.

1. Introduction

Over the last decade, multi-core processors have become the norm to provide high performance while staying within power constraints. As more cores were being integrated on the die, commercial operating systems are evolving to efficiently support the parallelism provided by multi-core processors [4]. In the meantime, ultra-low power small cores (e.g. Intel's Atom processor [3]) have emerged and show the potential to provide power-efficient performance in small form factor devices where extended battery life is crucial. As different types of cores are now available, the architectural options when designing a platform are also better. It also introduces the possibility for developing heterogeneous architectures that mix and match big and small cores on the same die to provide a range of power/performance capability. In addition to big and small cores, on-die integration of domain-specific accelerators for special-purpose functionality like graphics and media processing has also become wide-spread [5]. Future heterogeneous architecture research now needs to comprehend different types of cores as well as accelerators.

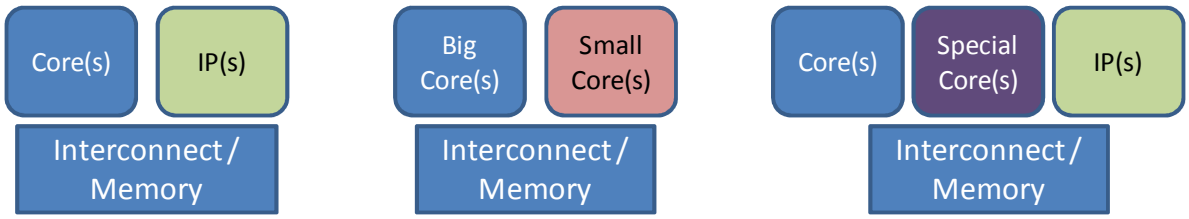
Heterogeneous architecture research [2, 6, 7, 8, 10, 11, 12, 13, 16] is challenging because it requires answers to questions such as (a) how many big and small cores should be supported in a platform? (b) what domain-specific accelerators should be introduced and how? (c) how should these heterogeneous processing elements be managed within the platform? (d) how should workload partitioning be done between these processing elements? (e) how should different applications be scheduled by the OS on these processing elements? and (f) how should applications be designed to cope with heterogeneity? For homogeneous multi-core processors, it is already known that simulation approaches are slow and daunting to perform detailed exploration. This is especially true for long-running experiments such as scheduling and migration of workloads across processing elements. For heterogeneous architectures, employing a similar approach becomes further challenging and therefore limiting. Our goal was to address this challenge for heterogeneous architecture exploration.

In this paper, we start by introducing the different types of heterogeneous architectures under consideration and then describe the research challenges in exploring the heterogeneous architecture options, OS implications and application mapping/development. To enable realistic exploration of these key research challenges, we introduce a QuickIA heterogeneous platform prototype. The QuickIA platform consists of multiple Intel cores (big and small) and includes FPGAs that can be used to synthesize accelerators and enable experimentation for domain-specific computation. We show three QuickIA prototype configurations and highlight their suitability for heterogeneous architecture research. Using simple case studies, we show the value of one of the QuickIA research vehicle configurations (asymmetric cores). We show that building and providing such a platform to architects and researchers allows them to study workload mapping, OS heuristics, performance/QoS and design space exploration. We believe this prototype platform is the first heterogeneous research vehicle and is currently being made available to selected academic groups for research purposes.

The rest of this paper is organized as follows. Section 2 introduces the different types of heterogeneous architectures being considered for exploration with this research vehicle. This section also describes the key research challenges. Section 3 introduces the QuickIA research vehicle for heterogeneous architecture exploration. Section 4 describes a few example case studies to highlight the types of experiments that can be done on such a QuickIA platform. Section 5 summarizes the contributions in this paper and outlines a direction for future work.

2. Heterogeneous Architecture Exploration

The design space for heterogeneous architecture is vast and needs careful consideration in terms of power, performance, programmability and flexibility. In this section, we classify a few heterogeneous architecture configurations under exploration and describe the research challenges.



(a) Core+IP Integration (b) Asymmetric Core Integration (c) Asymmetry and Specialization

Figure 1: Heterogeneous Architectures Under Exploration

2.1 Types of Heterogeneous Architectures

Figure 1 shows three types of heterogeneous architecture instances, which can be briefly described as follows:

(a) Core+IP Integration: One type of heterogeneous architecture (illustrated in Figure 1a) is to integrate multiple homogeneous cores with accelerators (also referred to as intellectual property (IP) blocks in the SoC domain). In this type of architecture, the IP block provides low power, high performance processing for specific domains. Examples include graphics, imaging, security, etc.

(b) Asymmetric Core Integration: Another type of heterogeneous architecture configuration consists of general purpose cores that are asymmetric in performance and power characteristics. An example would be the integration of big (out-of-order wide-issue core) and small (in-order narrow-issue core) cores targeted at providing performance or power efficiency when needed. The cores typically are based on the same ISA family, although could be from different generations.

(c) Asymmetry+Specialization: The last type of heterogeneous architecture configuration consists of asymmetric as well as specialized cores and accelerators. Here, the key difference is the introduction of specialized cores used for a specific purpose (hardware scheduling, management, domain-specific computation, etc).

As can be expected, performing simulation-based experimentation for these new (heterogeneous) class of architecture is not easy. While simulation-based experiments are still useful for micro-architecture experiments (to determine the right features in each core perhaps), enabling hardware prototyping that allows for experimentation across the different processing elements in these different domains is extremely useful since it provides long running, rapid exploration for runtime and application studies.

2.2 Key Research Challenges

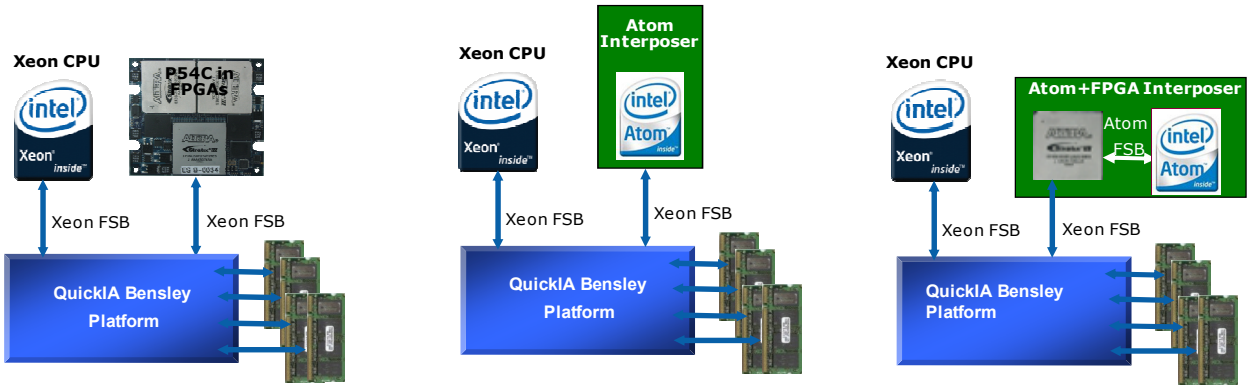
The research challenges in this area can be divided into three categories: (i) Architecture Exploration, (ii) OS Scheduling for Power/Performance/QoS, (iii) Application Partitioning and Mapping for Power/Performance/QoS and (iv) Heterogeneous programming models.

Architecture Exploration for Performance/Power/QoS: Architecture exploration for a heterogeneous platform can be quite wide and complex to explore. The first step for such experiments is running a chosen set of applications on the processing elements of interest and understanding the trade-offs in terms of performance, power and QoS. Deciding which parts of the application will run on the big or small core, which parts will be offloaded to an accelerator and identifying near-optimal configurations across a range of target applications is the focus here. The ability to run these applications on existing heterogeneous hardware with big and small cores allows for detailed performance profiling to be accomplished. In addition, availability of FPGA in the same platform can enable better experimentation of new accelerators of value.

OS Scheduling Implications: Today's OS schedulers are designed to manage homogeneity. They are unaware of the functional, performance and/or power differences between the cores (big and small cores for instance). Scheduling on these types of heterogeneous architectures can be made efficient from a performance and energy perspective if we can develop techniques that allow us to determine which application should be scheduled on which core (big or small). In order to explore OS scheduling heuristics for heterogeneous architectures, long running experiments are needed – the runs have to consist of many contexts to understand migration overheads as well as re-scheduling implications. A hardware platform prototype for heterogeneous architectures can provide rapid prototyping and speed up OS research and development.

Application Partitioning and Mapping: Once a heterogeneous platform is developed, emerging applications need to be developed such that they take the different processing elements into consideration. For example, phases in an application that require high performance should be targeted to a big core and phases that do not require performance should be targeted to small cores. Similarly, specific phases that belong to the same domain as the accelerators available in the platform should be written such that they can take advantage of it. A hardware prototype platform enables rapid application partitioning experiments.

Programming Model Implications: Programming the heterogeneous parallel platform poses several challenges – especially if we are dealing with accelerators and functional ISA differences. Today, the programmer accesses accelerators in the platform, treating them as devices. One drawback of this approach is the memory model used in the device domain. The programmer needs to know the custom hardware interfaces and need to communicate with the hardware in physical address domain through complex and inefficient page pinning and virtual to physical address translations. Another significant challenge is the portability of software written for one heterogeneous architecture instance versus another. It is important to implement built-in support for portability by ensuring that any code that is hardware accelerated can either run directly on the hardware instantiation of that accelerator or run seamlessly as software on a general-purpose or specialized core. This requires support in the hardware for detecting existing accelerators as well as support in the software to implement multiple execution paths for different elements. Experiments on heterogeneous platforms allow for understanding the performance and programmability limitations of traditional models and allow for identifying where better solutions are needed.



(a) QIA-HC1: Xeon+FPGA *(b) QIA-HC2: Xeon+Atom* *(c) QIA-HC3: Xeon+Atom+FPGA*

Figure 2. QuickIA Research Vehicle: Prototype Configuration for Heterogeneous Architecture Exploration

3. QuickIA: A Hetero Prototype Platform

The QuickIA research platform is built and designed to provide the necessary heterogeneous components i.e. big cores, small cores and fixed function accelerators all on a single platform thereby allowing workload analysis in a real heterogeneous environment. It should be kept in mind that a hardware prototype obviously has some rigid components, and therefore it should not be considered a full replacement to simulation or other approaches to emulation.

3.1 QuickIA Hardware

The QuickIA platform is based on a dual socket Xeon 5400 series server. These two CPU sockets are connected via the Intel Front Side Bus (FSB) to the memory controller hub on the platform. Using this as a baseline, three configurations of the platform were created as shown in Figure 2. These configurations allow both the sockets to be fully cache coherent (as necessary) with full access to the platform services like memory and I/O. By using the Xeon, Atom and FPGA we are able to create functional, and performance asymmetry making it ideally suited for heterogeneous investigations. The three specific configurations will be labeled and described below as follows:

1. **QIA-HC1**: Socket 0 (Xeon), Socket 1 (FPGA to prototype different accelerators and cores of interest)
2. **QIA-HC2**: Socket 0 (Xeon), Socket 1 (Atom N330)
3. **QIA-HC3**: Socket 0 (Xeon), Socket 1 (Atom N330 + FPGA)

QIA-HC1 is a hetero prototype configuration in which one socket is a Xeon 5450 CPU and other socket is a FPGA device. The FPGA can be used to synthesize accelerators for "Core+IP" research. To get started, we implemented a soft core (P54C) functionally in the FPGA [17]. Although this soft core is a very old Pentium core configuration, we were able to expose it to the OS and make it behave like actual cores in real silicon, only difference being the slow operational speed. Experiments using this configuration include changing cache size, add/modify instructions, and modifying the CPUID so as to "spoof" the capabilities of the soft cores to the OS. This configuration is not good for benchmarking applications since the soft cores run very slow (~50-75MHz) when compared to the Xeon core (3GHz). The FPGA module can support upto four P54C cores.

QIA-HC2 is a more balanced configuration that has a real Xeon 5450 CPU in one socket and a real Atom N330 CPU in the other socket. This configuration was designed to run real workloads and collect data since both the big and small cores are implemented as real CPUs. All experimental data presented in this paper is based on this configuration. In the QIA-HC2 configuration, we emulate a heterogeneous platform in which the big and small cores run at the same core frequency and also have comparable cache hierarchies. By default, the Xeon and the Atom CPUs have very different FSB and cache configurations making it unsuitable to use as is. By modifying the motherboard, the BIOS settings, and the processor sockets, both configurations were made as close to each other as possible.

Table 1: QIA-HC2 Configuration of Xeon5450+AtomN330

Feature	Xeon 5450 (Harpertown)	Atom N330 (Silverthorne)
# of Cores	2	2
Core speed	1.60GHz	1.60GHz
FSB speed	533MHz	533Mhz
L1 Instr Cache	32KB	32KB
L1 Data Cache	32KB	24KB
L2 Cache	2MB/2cores (1MB/core)	512KB/core
Memory Addressability	36bits PA	36bits PA
HT	Not Available	OFF
Prefetchers	OFF	OFF
ISA	Upto SSE4	Upto SSE3
C States	ON	Not Available

QIA-HC3 is designed to emulate specialized cores or tightly-coupled accelerators in addition to the big and small cores. This configuration uses a special module that has an Atom N330 CPU connected to a FPGA which in turn is connected to the host platform Front Side Bus. This allows the Atom and the accelerators direct access to the memory and also allows close coupling between the Atom and FPGA. The Atom CPU is visible to the OS as if it were directly connected to the socket as in QIA-

HC2. This platform is most suited for workloads that require closely coupled accelerators like crypto engines, packet inspection engines etc.

3.2 QuickIA Software Support

In this section, we introduce the various issues that system software and application software needs to consider when running on QuickIA or QuickIA-like x86 systems. These are a result of our experiences in bootstrapping and benchmarking the QuickIA platform while conducting heterogeneous asymmetric multi-processor systems (ASMP) research. As is evident, a lot of these problems arise because current state-of-the-art systems software assumes homogeneous SMPs. Our experience has been mostly with the Linux operating system. However, most of these issues are likely to be encountered when using other operating systems as well. The following is a list of issues that a typical QuickIA user is likely to face:

- The Linux kernel uses non-architectural features such as model–specific registers (MSR). One example of such a use is the last branch record (LBR) feature in the *perf* subsystem. When these registers need to be reset, the addresses of these registers are model–specific, and differ between the E5450 and N330 processors in the QuickIA platform. The Linux kernel, assumes that the registers are same, thus resulting in machines crashing in strange ways.
- Modern system software like *eglibc* provides several implementations for frequently used performance-critical functions like *memcpy*. However, the core on which the 'cpuid' instruction is not necessarily the core on which the rest of the process runs. This can cause faults in the early user-space which can cause re-writing certain parts of core-system software or leave out the performance on the table. The *eglibc* library has an SSE 4.1 optimized implementation of the above mentioned function. We encountered one such critical issue with *udevd*. To work around this, we used the *eglibc* dynamic loader's pre-load mechanisms to always load a Lowest Common Denominator version of *memcpy* function.
- The 'alternatives' mechanism allows Linux to take advantage of the advanced feature-set of newer processor while still supporting older hardware without recompilation. During build–time, the kernel stores two (or more) implementations of a given functionality. After discovering the platform capabilities at boot time, the alternative implementations are patched into the kernel image appropriately. Examples of such patching in the Linux kernel include memory barriers, saving and restoring the floating point state, SMP locking primitives, atomic operations, operations on bitmaps and pre-fetch hints. Indeed, with heterogeneous capabilities only the baseline is guaranteed to work correctly. We disabled this mechanism on the Linux kernel for QuickIA.
- During period of inactivity, the processor progressively shuts off various components and saves power and thus reduces energy consumption. The *cpuidle* driver handles the idle–state management in the Linux kernel. The device driver developers can provide hints to the idle management subsystem about the tolerable latencies in a given device state. The *cpuidle* driver ensures that there is only one idle loop algorithm for all the cores in the system. However, each of the cores can be in a different state at any given point of time. The idle states are initialized only once for the BSP and it is assumed that the same hold true for all the cores. The Linux kernel that ran on our QuickIA platform could use only two idle-states (C0 and C1). Ideally, one would expect that the operating system would use per–cpu idle loops that are optimized for a given core architecture for maximal power savings and appropriate entry and exit latencies. For a heterogeneous system, Linux should have a *cpuidle* driver per–cpu which could be different and uses the C–states that are specific to the core.
- DVFS allows for a mechanism to save power at lower clock frequencies (at low voltages) that result in lower power consumption. DVFS operating curves for the cores in a QuickIA-like system are different. In the Linux kernel, a single driver provides the mechanism for driving the various cores in the platform to different P–states. The governor mechanisms in the Linux kernel assume that the DVFS operating curves of all the cores in a system are the same. On the QuickIA system this is not a correct assumption because all the cores have different sets of possible P-states. We disabled the DVFS functionality for our initial set of experiments on the QuickIA system.
- Intel Architecture processors implement a mechanism that can detect and report hardware errors called the Machine Check Architecture (MCA). To this extent they consist of several error–reporting *register banks*. Each bank is associated with one or more hardware units. The QuickIA platform consisted of cores that supported different numbers of MCA banks. The Linux kernel assumes that the banks discovered in the BSP are applicable to all the APs. In our software stack that ran on the QuickIA system, we disabled the machine check functionality.
- The Linux kernel's hardware-discovery process enumerates the bootstrap processor's features and logically ANDs these features as the discovery process happens on the application processors. This result in the lowest common denominator subset of features exposed to the software. This enumeration mechanism prevents the use of the performance features of the big-cores. In our early enabling we used the lowest common denominator approach which left a lot of performance on the table. Subsequently, we enabled per-processor feature flags to be present and used mechanisms like fault-and-migrate to cope with feature heterogeneity.

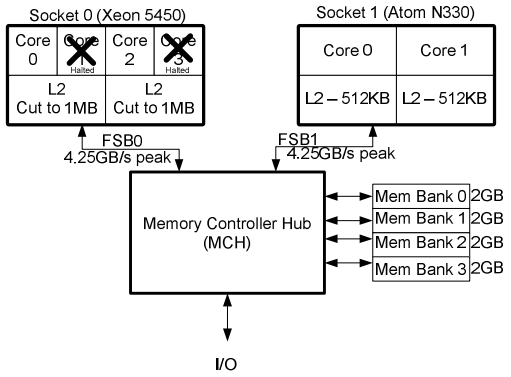


Figure 3: QuickIA-HC2 configuration used for case studies

4. Illustrative Case Studies on QuickIA (HC2)

In this section, we will now describe a few example case studies that illustrate the value of the heterogeneous architecture prototype. We focus our attention on the QuickIA-HC2 platform for this purpose. Figure 3 shows the configuration and Table 1 previously showed the key parameters.

4.1 Baselining Application Performance

Comparisons of big and small cores are generally flawed because they are done on standalone systems which have differing platform configurations. Our QuickIA-HC2 platform prototype enables both cores at identical frequency and platform configuration in order to allow for more consistent comparisons. Figure 4 shows the performance difference between running a sample set of applications on big and small cores. It should be noted that these applications were not specifically optimized for one purpose or the other, so the comparison provided here should not be treated as benchmark results by any means.

Figure 4 clearly shows that the performance ratio between big and small cores has a wide spectrum from 3.4X down to 1.2X. This clearly indicates that if a heterogeneous platform is designed with big and small cores, there is opportunity to optimize the platform throughput and individual performance of an application based on speedup observed. This also implies that hardware or software mechanisms that provide more insight into the behavior of the workload are valuable since this allows for application scheduling to be biased towards a small core or a big core. More research along these lines is on-going and such a prototyping platform offers great value in dynamically profiling the behavior of individual applications to perform these studies.

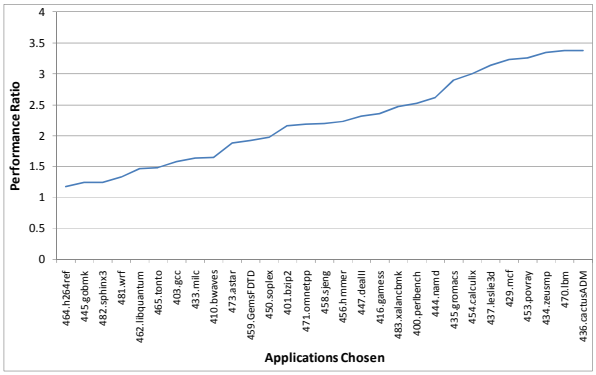


Figure 4. Application Performance Ratio (Big/Small)

4.2 Hetero Throughput Improvement Potential

In the last section, we showed that different applications show different speedup ratios on big vs. small cores available in a heterogeneous platform. To illustrate the throughput improvement potential based on this knowledge, we performed simple multi-programming experiments.

We chose three groups of four applications each and ran them on the four cores configured (2 big cores and 2 small cores). The first group chose two applications from each end of the x-axis on Figure 4. The second group chose applications from the middle, and the third chose at random. We ran the three groups with two types of affinity. The default case is labeled "w/o scheduling" and optimized case is labeled as "w/ scheduling". In the "w/o scheduling" case, the four applications are randomly scheduled in the four cores, whereas in the "w/ scheduling" case, two applications that have smaller speedup ratio (based on Figure 4) are scheduled on the small cores, and the other two are scheduled on the big cores.

Figure 5 shows the moderate performance benefits as a result of this simple policy. The y-axis is the sum of IPC of each application normalized to the IPC when it is running alone on a big core. As shown in Figure 5, the overall improvements (height of each bar) ranges from 5% to 36% with simple improvements in affinitization. In addition, there are per application improvements which can be noted by comparing the sections across the two bars for each application. In one case (group2), one application also reduces in performance, so trade-offs such as those should also be kept in mind. The purpose of this experiment was just to highlight the improvement possible by efficient mapping of applications. In a later section we will show how OS heuristics for such purposes can be designed and evaluated on this platform.

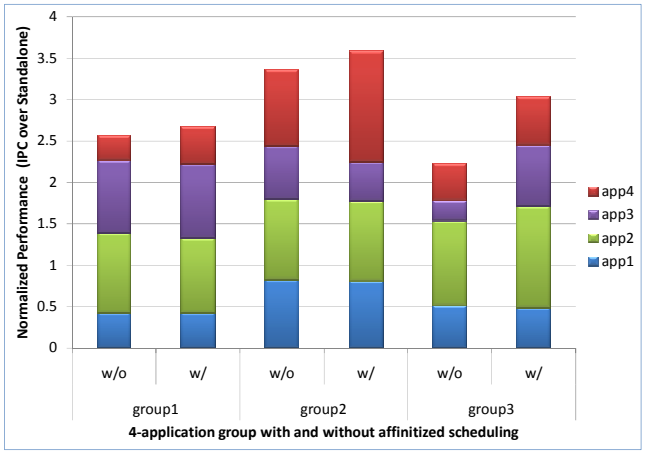


Figure 5. Throughput and Individual Application Performance

4.3 Hetero QoS Experiments

With differing cores providing different levels of performance, it is possible to provide differentiated performance to applications based on their priority or SLA (Service Level Agreement). To highlight this, we ran simple experiments with affinitization to show the improvements in QoS that are possible. We chose three groups of 8 applications each. Within each group, we ran two experiments: w/o QoS and w/ QoS. In the experiment w/o QoS, we essentially allowed all applications to be scheduled on any of the four cores (2 big and 2 small cores). In the experiment w/ QoS, we chose an application (testapp1) as a high priority application, and affinitized it to a big core. All other applications were affinitized to the remaining cores. We then chose another application (testapp2) and do the same experiments.

Figure 6 shows the speedup of the application when it is treated as high priority application (w/ QoS) compared to the base case (w/o QoS). As can be observed from the figure, the benefits in application performance ranges from 3X to 4.5X by avoiding contention with other applications and employing a big core for this high priority application. While this experiment was very targeted, heuristics that achieve similar QoS techniques can be developed and evaluated on this platform.

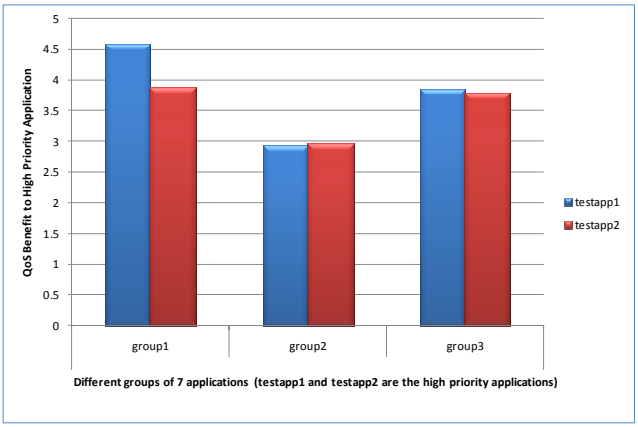


Figure 6. QoS Benefits for High Priority Applications

4.4 Hetero Core Space Exploration

When heterogeneous architecture is designed, it has more design options than homogenous architecture. For example, in addition to the tradeoff of core space vs. cache space on die, we have to consider the number of big cores and the number of small cores that should be integrated on die. To explore the core design space, we do some experiments to compare one architecture with 2 big cores and the other with 1 big core along with 2 small cores assuming the two small cores have the similar die space as one big core. Table two lists the four groups of applications that we choose to run. The first application is always running on the first big core, and the last two applications are either running on the other big core or two small cores. Figure 7 shows the total IPC on these two architecture.

Table 2. Four groups of applications

group1	CACTUSADM,H264REF,GOBMK
group2	CACTUSADM,LIBQUANTUM,ASTAR
group3	MCF,H264REF,SOPLEX
group4	OMNETPP,ASTAR,SOPLEX

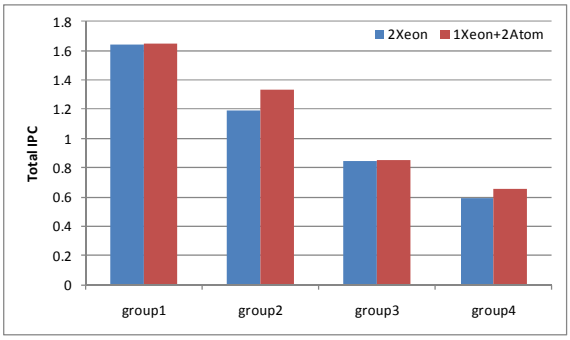


Figure 7. Total IPC on 2 Xeon vs. 1 Xeon and 2 Atoms

We can see that when we replace one big core with two small cores, there is no significant difference in total IPC for group1 and group3. However the IPC is increased by 12% and 10% for group2 and group4 respectively. This experiment just shows one example of core space exploration that we can perform on the hetero platform to help design efficient heterogeneous architecture for various workloads.

4.5 Hetero OS Scheduling Heuristics

With different cores providing different levels of performance, it is possible to dynamically schedule processes which are best suited to a particular core type. Such scheduling algorithms can be implemented in a operating system to do make online decisions or maximizing power, performance or a combination of both. Further, it is possible to create more complicated schemes which involve solving a discrete-time dynamic optimization problem (with various design parameters as constraints)to predict the best schedule at a given time instant. In previous work [8] we have proposed and implemented simple algorithms that can leverage a heterogeneous ASMP system to maximize performance. We repeated the fastest-core-first experiments on the QuickIA system to verify if those algorithms still hold up to their promise. This algorithm tries to keep the big cores always busy. First, when making a placement decision it will always prefer a fast core. Secondly, when a faster core finishes its work, it will pull running tasks on the smaller cores to keep the faster core busy. Figure 8 shows the speedups gained by FCF when running 4 copies of each of the integer SPEC benchmarks on a 4 core QuickIA system with two big and two small cores. The benchmark's runtime is the geo-mean of all the copies' runtimes. As can be seen, the speedup can be very significant. This illustrates the fact that there is a lot of potential for intelligent OS scheduling in ASMP systems.

Furthermore, the scheduler can keep an account of a particular task's runtime architectural characteristics (like IPC, stall metrics, etc) and make intelligent decisions based on it. We are currently working on the correct set of heuristics which will provide the most optimal schedule at each scheduling decision-point.

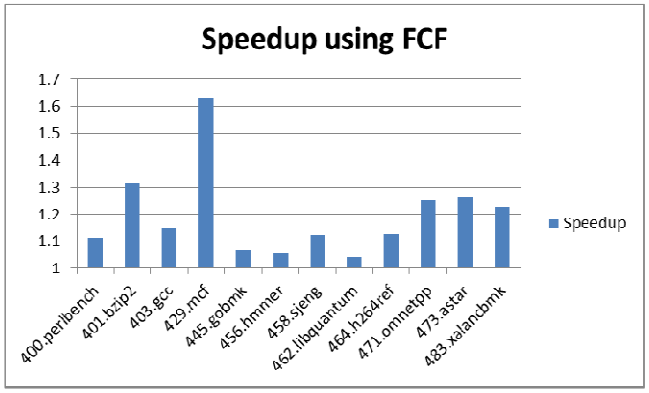


Figure 8. Speedup using FCF vis-à-vis stock Linux scheduler

5. Summary and Future Work

The potential transition to heterogeneous architectures fuels architecture research by introducing both opportunities in power/performance and challenges in designing and enabling them in software. Exploration of heterogeneous architectures requires prototyping platforms since the interaction between applications, OS and heterogeneous hardware needs to be explored. In this paper, we introduced the QuickIA series of heterogeneous platform prototypes that serve as research vehicles for architects, OS researchers and applications programmers.

We described the three QuickIA platform configurations (consisting of big cores, small cores and FPGA) that have already been developed and tested. We focused on the heterogeneous core configuration and highlighted a few example case studies that

point to further research needed and the value of such a research vehicle. This QuickIA configuration is being made available to a limited set of university research groups to help academic research as well as advance heterogeneous computing research in the architecture community. Researchers receiving the QuickIA platform also receive the heteroOS kernel source code under the GPL. We believe that this QuickIA platform is a first of its kind and will facilitate a growth in exploration and compelling results on this important and growing area of research.

The results and discussion in this paper do not indicate any product ideas inside Intel Corporation. It also is not representative of the performance of production systems designed using the CPUs discussed in our prototype

References

[1] D. Anderson, et al, "FAWN: A Fast Array of Wimpy Nodes," Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP 2009), 2009

[2] J. Chen and L. K. John. Efficient Program Scheduling for Heterogeneous Multi-core Processors. In Proceedings of Design Automation Conference, 2009.

[3] Intel® Atom™ Processor. http://www.intel.com/technology/atom/

[4] Intel® Core™2 Duo Processor. http://www.intel.com/products/processor/core2duo/index.htm

[5] Intel's Next-Generation Handheld Platform ("Moorestown"), http://www.intel.com/pressroom/archive/reference/Moorestown_backgrounder.pdf

[6] R. Kumar, K. I. Farkas, N.P. Jouppi, P. Ranganathan, D.M. Tullsen., "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction," In Proceedings of the 36th International Symposium on Microarchitecture.

[7] R. Kumar, D.M. Tullsen, P. Ranganathan, N.P. Jouppi, K. I. Farkas. "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance" In Proceedings of the 31st International Symposium on Computer Architecture.

[8] T. Li, P. Brett, et al. Operating System Support for Overlapping-ISA Heterogeneous Multi-core Architectures. In Proceedings of High Performance Computer Architecture, 2010.

[9] V. Reddy, B. Lee, T. Chimbi, K. Vaid, "Web search using mobile cores: Quantifying and mitigating the price of efficiency" ACM SIGARCH Computer Architecture News, 2010.

[10] D Shelepov, et al, "HASS: A Scheduler for Heterogeneous Multicore Systems," Operating Systems Review, vol. 43, issue 2, (Special Issue on the Interaction among the OS, Compilers, and Multicore Processors), April 2009.

[11] S. Ghiasi and D. Grunwald. Aide de camp: Asymmetric dualcore design for power and energy reduction. In University of Colorado Technical Report CU-CS-964-03, 2003.

[12] R. Kumar, D.M. Tullsen, N.P. Jouppi. Core Architecture Optimization for Heterogeneous Chip Multiprocessors. In proceedings of Parallel Architectures and Compilation Techniques, 2006.

[13] M. Becchi and P. Crowley. Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures. In Computing Frontiers, 2006.

[14] SM10000 High Density, Low Power Server. http://www.seamicro.com/sites/default/files/SM10000DS.pdf

[15] Standard Performance Evaluation Corporation. SPEC Benchmark Suite. http://www.spec.org

[16] D. P. Gulati, C. Kim, S. Sethumadhavan, S. W. Keckler, and D. Burger. Multitasking Workload Scheduling on Flexible Core Chip Multiprocessors. ACM SIGARCH Computer Architecture News, Vol. 36, Issue 2, May 2008

[17] Qigang Wang, Rolf Kassa, Wenbo Shen, Nelson Ijih, Bhushan Chitlur, Michael Konow, Dong Liu, Arthur Sheiman, Prabhat Gupta, "An FPGA Based Hybrid Processor Emulation Platform," In Proceedings of FPL'2010. pp.25~30