Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2010-008 February 11, 2010

Core Count vs Cache Size for Manycore Architectures in the Cloud

David Wentzlaff, Nathan Beckmann, Jason Miller, and Anant Agarwal

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Core Count vs Cache Size for Manycore Architectures in the Cloud

David Wentzlaff, Nathan Beckmann, Jason Miller, and Anant Agarwal
CSAIL, Massachusetts Institute of Technology
Cambridge, MA 02139
{wentzlaf, beckmann, jasonm, agarwal}@csail.mit.edu

***Abstract*—The number of cores which fit on a single chip is growing at an exponential rate while off-chip main memory bandwidth is growing at a linear rate at best. This core count to off-chip bandwidth disparity causes per-core memory bandwidth to decrease as process technology advances. Continuing per-core off-chip bandwidth reduction will cause multicore and manycore chip architects to rethink the optimal grain size of a core and the on-chip cache configuration in order to save main memory bandwidth. This work introduces an analytic model to study the tradeoffs of utilizing increased chip area for larger caches versus more cores. We focus this study on constructing manycore architectures well suited for the emerging application space of cloud computing where many independent applications are consolidated onto a single chip. This cloud computing application mix favors small, power-efficient cores. The model is exhaustively evaluated across a large range of cache and core-count configurations utilizing SPEC Int 2000 miss rates and CACTI timing and area models to determine the optimal cache configurations and the number of cores across four process nodes. The model maximizes aggregate computational throughput and is applied to SRAM and logic process DRAM caches. As an example, our study demonstrates that the optimal manycore configuration in the 32nm node for a 200mm^2 die uses on the order of 158 cores, with each core containing a 64KB L1I cache, a 16KB L1D cache, and a 1MB L2 embedded-DRAM cache. This study finds that the optimal cache size will continue to grow as process technology advances, but the tradeoff between more cores and larger caches is a complex tradeoff in the face of limited off-chip bandwidth and the non-linearities of cache miss rates and memory controller queuing delay.**

I. Introduction

Multicore and manycore architectures containing anywhere from four cores to 64 cores are commonplace today [11], [19], [20], [12], [8]. As technology advances, the core count has the potential to increase by a factor of two every generation, even with a fixed chip area. Architects must however weigh several factors in order to determine the right number of cores and the composition of each core. In the absence of other constraints, increasing the number of cores can increase performance proportionally for throughput oriented workloads such as cloud computing, or for applications with high degrees of parallelism. Several architectural constraints limit the benefits of the increasing number of cores. Chief among these constraints is the bandwidth limits of the interconnect architecture that is used to connect the cores, and the bandwidth limits to off-chip DRAM memory.

To address the on-chip interconnect challenge, multicores are replacing the traditional bus architecture with point-to-point interconnects such as crossbars [11], [1], [19] and rings [12], [18]. Even these do not scale to more than 10 or 20 cores, so manycore architectures have begun to adopt meshes [23], which can scale to a few hundred cores.

The off-chip DRAM bandwidth challenge is a much harder problem with no immediate solution in sight. Although growth in the number of cores on a chip is exponential according to Moore's Law, off-chip memory bandwidth is growing linearly at best. Growth in external DRAM bandwidth will come from a combination of gradual increases in the pin count and the frequency at which the pins are driven. This imbalance in the growth rates of core count and external bandwidth will cause a decrease in the external bandwidth available to each core as technology advances. Technologies such as stacked DRAM will offer a one-time boost in the available external DRAM bandwidth, but will not provide a permanent solution to the problem.

To mitigate the external DRAM bandwidth bottleneck, multicore architectures will integrate increasing amounts of cache on each core.[1] Furthermore, integrating dense DRAM on the multicore die as an additional level of cache will become increasingly fruitful. Increasing the amount of on-chip cache per core (whether implemented as SRAM or DRAM) will reduce the bandwidth demand imposed on off-chip DRAM. However, it will also reduce the number of cores that can be integrated on a die of fixed area.

Given the large number of design parameters in multicore and manycore designs, multicore architects need aids in determining the appropriate values for parameters such as cache sizes, number of cores, cache organization, number of execution units, and on-chip network configurations. In this work, we derive an analytic model to compare the tradeoff between different cache configurations within a core and number of cores for a fixed die area. We build a framework for homogeneous multicores around our analytic model using SPEC Int 2000 cache miss rates and the CACTI 5.3 [22] cache and DRAM timing and area models to determine optimal cache size and core count configurations across four process technology nodes. This work studies candidate configurations which use SRAM or embedded DRAM for L2 caches.

In this study, we utilize the growing industry of Cloud computing as a motivational application area for this work. We believe that Cloud computing and Infrastructure as a

[1] In recent times, we have seen a *decrease* in the amount of cache per core in order to make space for more cores. This decrease simply reverses the uniprocessor trend of integrating increasing amounts of cache per chip in the absence of any other way to use exponentially increasing transistors. This trend will be reversed yet again as the increasing number of cores hits against the off-chip DRAM bandwidth constraint.

Service will be a killer application for future manycore processors. Infrastructure as a Service providers such as Amazon's EC2 [2] and Platform as a Service services such as Google's App Engine [5] today utilize large amounts of computation and in the future could grow to a tremendous amount of parallel computation. Because of this huge potential we have decided to choose an application mix which is comprised of many independent sequential applications being aggregated onto future manycore processors. This models the data center aggregation workload endemic of Infrastructure as a Service and Platform as a Service workloads.

The model presented in this work can easily be configured to model a wide range of processor types, sizes, and configurations. For our cloud application motivated study we focus on power-conscious processor configurations. Cloud computing and data center aggregation optimizes for a throughput oriented workload and are motivated by providing the highest throughput for the lowest power. As such this study investigates small low-power in-order, two to three way issue processors such as the Tilera core or MIPS32 24k core in contrast to large out-of-order superscalar processors which are less well area and power efficient for throughput oriented computations.

From this study, we find that the optimal per-core cache size grows as process technology advances. The optimal core count also increases, but the tradeoff is more subtle as some of the increased area provided by process scaling is better used for cache in the face of off-chip bandwidth constraints. We find that the area density benefits of using embedded DRAM as an L2 cache always outweigh the latency overhead when compared to SRAM.

This paper begins by deriving an analytic model of multicore memory performance in Section II. Section III describes the experimental setup, Section IV details the results found. Section V and Section VI describe related and future work, and finally we conclude.

II. An Analytic Memory Model of Multicore Processors

We present an analytic model of multicore performance, with emphasis on the memory system. Our model describes a processor with N homogeneous cores. Each core has conventional L1 instruction and data caches backed by an L2 cache. The L2 is connected via the on-chip network to main memory off chip. The model is constrained on the area of a chip and off-chip bandwidth. Our model allows for parameters from many different technologies and process nodes to be evaluated and compared in the same framework. Table I lists the parameters to the model.

Name	Description
CPI_0	Cycles per instruction of non-memory operations.
f_{mem}	Frequency of memory references (data references per instruction).
N	Number of cores.
m_i	Miss rate in level i of the memory hierarchy.
t_i	Access time to level i of the memory hierarchy.
ℓ_i	The cache line size of level i of the memory hierarchy.
A	Area of chip.
A_{core}	Area of a single core, not including caches.
A_i	Area of level i of the memory hierarchy.
B	Off-chip bandwidth, in bytes per second.
f_{core}	Clock frequency of each core.
M	The number of memory controllers.
w_{net}	The width of the on-chip network.

TABLE I
Model parameters.

A. Derivation of the Model.

1) Cycles Per Instruction: We begin by modeling the cycles per instruction (CPI) of a single core. The foundation of the model is standard equations for modeling stalls from misses at different levels of the memory hierarchy [13]. In addition, the model accounts for network latency and off-chip contention with the values T_N and T_Q.

The basic form of the CPI equation is:

$$\begin{aligned}\text{CPI} = {} & \text{CPI of non-memory instructions} + \text{I-cache miss penalty} + \text{CPI of} \\ & + \text{Network congestion} + \text{Memory congestion}\end{aligned}$$

This is expressed in the parameters above as:

$$\begin{aligned}\text{CPI} = {} & \text{CPI}_0(1 - f_{\text{mem}}) + m_{\text{IL1}}(t_{\text{IL1}} + t_{\text{L2}} + m_{\text{L2}}t_{\text{mem}}) \\ & + f_{\text{mem}}(t_{\text{DL1}} + m_{\text{DL1}}(t_{\text{L2}} + m_{\text{L2}}t_{\text{mem}})) \\ & + (m_{\text{IL1}} + f_{\text{mem}}m_{\text{DL1}})m_{\text{L2}}(T_N + T_Q)\end{aligned} \quad (1)$$

2) Modeling Off-Chip Contention.: We model contention for off-chip bandwidth as a bus using a standard M/D/1 queueing model. Let λ be the mean arrival rate of messages in the queue, modeled as a Poisson process.[2] Let μ be the service rate, a constant. Define ρ, the line utilization, as $\rho = \frac{\lambda}{\mu}$. Then the queuing delay, W, is derived to be,

$$W = \frac{\rho}{2\mu(1-\rho)} \quad (2)$$

In our case, the arrival rate in the queue is the rate of off-chip memory references. This can be expressed in terms of the misses to main memory per core per cycle, λ_{core}. λ_{core} is derived using the misses to main memory per instruction, divided by the cycles per instruction.

$$\lambda_{\text{core}} = \frac{1}{\text{CPI}}\left(m_{\text{IL1}} + f_{\text{mem}}m_{\text{DL1}}\right)m_{\text{L2}}$$

There are N cores and we assume that memory references are divided evenly among M memory controllers, so $\lambda = \frac{N\lambda_{\text{core}}}{M}$.

The service rate is the number of cycles required by the bus to transfer the request. This depends on the off-chip memory

[2]Although memory accesses from individual cores are bursty, with many independent cores the bursts are smoothed, and we believe Poisson is a good model of system behavior. In the same setting memory accesses will be distributed uniformly, assuming memory addresses are striped across memory controllers.

bandwidth at each memory controller and the size of each request. With w_{bus} as the width of the off-chip bus and f_{bus} as the frequency of the bus,

$$\mu = \frac{w_{\text{bus}} \frac{f_{\text{bus}}}{f_{\text{core}}}}{\ell_{\text{L2}}} = \frac{B/M}{\ell_{\text{L2}} f_{\text{core}}}$$

To derive the line utilization, it is useful to define Λ, the byte-rate of memory requests off chip.

$$\rho = \frac{\lambda}{\mu} = \frac{N \ell_{\text{L2}} f_{\text{core}} \lambda_{\text{core}}}{B} = \frac{\Lambda}{B}$$

Thus, the line utilization is the ratio of the off-chip request rate to the off-chip bandwidth, as expected.

T_Q can be computed directly from Eq. 2 with these parameters. Define Λ_0 as the request rate with a nominal CPI of 1, $\Lambda_0 = \Lambda \cdot \text{CPI}$.

$$T_Q = \frac{\Lambda}{2\mu(B - \Lambda)} = \frac{\Lambda_0}{2\mu(B \cdot \text{CPI} - \Lambda_0)} \tag{3}$$

Now CPI appears on both sides of the earlier CPI equation (Eq. 1). Intuitively, this is because the request rate to main memory is self-limiting. As off-chip bandwidth becomes a limiting factor in performance, CPI of each processor will increase from queueing delay. This in turn reduces the request rate to main memory from each processor.

We now solve for CPI. For clarity, define CPI_{base} as the CPI without off-chip contention. This is essentially the CPI of each processor with infinite off-chip bandwidth.

$$\text{CPI}_{\text{base}} = \text{CPI} - (m_{\text{IL1}} + f_{\text{mem}} m_{\text{DL1}}) m_{\text{L2}} T_Q$$

Substituting Eq. 3 into Eq. 1, we get a quadratic formula in CPI. Solving for CPI,

$$\text{CPI} = \frac{\text{CPI}_{\text{base}}}{2} + \frac{\Lambda_0}{2B} \pm \frac{\sqrt{\frac{2M\Lambda_0^2}{N} + (\Lambda_0 - B \cdot \text{CPI}_{\text{base}})^2}}{2B} \tag{4}$$

Note that the "+" form of the above equation should be used, because CPI increases as Λ_0 increases.

3) Modeling Network Latency.: Network latency is modeled by the number of hops needed for a packet to reach a memory controller. Our model assumes the N processors are arranged in a square two-dimensional mesh with dimension-order wormhole routing and single-cycle latency per hop. There are four memory controllers at the midpoint of each edge of the chip, and memory references are uniformly distributed among them. The average number of network hops for any core is twice the average distance in each dimension, or,

$$2 \cdot \frac{1}{4} \cdot \left(2 \cdot \frac{\sqrt{N}}{2} + 2 \cdot \frac{\sqrt{N}}{4} \right) = \frac{3\sqrt{N}}{4}$$

Our model also must account for the length of the message. There is additional serialization latency for each flit past the first. Each message is $\frac{\ell_{\text{L2}}}{w_{\text{net}}}$ flits long. This gives a network latency of,

$$T_N = \frac{3}{4}\sqrt{N} + \left(\frac{\ell_{\text{L2}}}{w_{\text{net}}} - 1 \right) \tag{5}$$

Finally, our model assumes that network congestion (queueing delay) will not play a significant factor in the latency of memory requests. This is justified because congestion only becomes a significant factor when utilization of the network is high. Generally speaking, off-chip memory bandwidth is the bottleneck, not on-chip bandwidth — the network will easily support the off-chip bandwidth, which is the limiting factor in overall performance (Section II-B).

However, it is simple to model congestion within this framework using analytical mesh models. Using the model in [7], the average hops in each dimension is $k_d = \frac{3\sqrt{N}}{8}$. This gives network utilization of,

$$\rho_{\text{net}} = \lambda_{\text{core}} \left(\frac{\ell_{\text{L2}}}{w_{\text{net}}} \right) k_d$$

Which yields additional congestion delay of,

$$\frac{3}{2} \cdot \frac{\rho_{\text{net}} \frac{\ell_{\text{L2}}}{w_{\text{net}}}}{(1 - \rho_{\text{net}})} \cdot \frac{(k_d - 1)}{k_d^2}$$

The only complication is that λ_{core} still depends on CPI, so the network congestion does as well. The same method used in the last section to solve for CPI with off-chip contention can be used to solve for CPI.

We repeated all experiments using the above network congestion model with a variety of network sizes, both constrained and over-provisioned. Iterative numerical methods were used to solve for the correct value of CPI. We found that in all cases, the inclusion of network congestion made no significant change in results. Only the most restrictive network configurations produced any change at all in Table IV, including the Chip IPC values down to the second decimal place. This justifies excluding network congestion from the model. In particular, its inclusion yields a more complicated equation for CPI that we believe clouds the main performance trends.

4) Chip IPC.: Our overall fitness metric is IPC for the chip as a whole. This is simply,

$$\text{IPC}_{\text{chip}} = \frac{N}{\text{CPI}} \tag{6}$$

N can be computed from the area of the chip and the area of a core. Using a separate cache model (Section III), one can also compute the area of the memory subsystem per tile. This gives,

$$N = \frac{A}{A_{\text{core}} + A_{\text{cache}}} \tag{7}$$

Where, $A_{\text{cache}} = A_{\text{IL1}} + A_{\text{DL1}} + A_{\text{L2}}$.

B. Limits of Performance.

We now consider the approximate behavior of Eq. 4 as off-chip bandwidth is exceeded. For reasonable values of M (say, 4), it is safe to eliminate the M term in Eq. 4 to observe its approximate behavior. When this is done, the equation becomes,

$$\text{CPI} \approx \begin{cases} \text{CPI}_{\text{base}} & \text{if } \Lambda_0 \leq B \cdot \text{CPI}_{\text{base}} \\ \frac{\Lambda_0}{B} = \rho_0 & \text{otherwise} \end{cases} \tag{8}$$

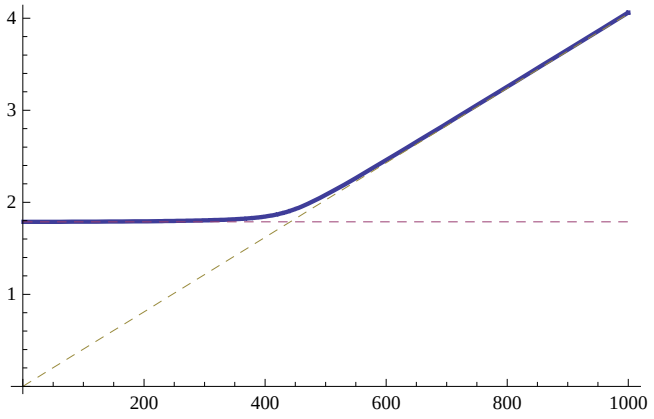


Fig. 1. CPI (Eq. 4) versus approximation (Eq. 8). Parameters are taken from those used to generate final results.

The point $\Lambda_0 = B \cdot \text{CPI}_{\text{base}}$ corresponds to $\Lambda|_{\text{CPI}=\text{CPI}_{\text{base}}} = B$. So the behavior changes when the request rate of the processor exceeds off-chip bandwidth. If the processor does not exceed off-chip bandwidth, CPI is approximately the base CPI (no impact from bus contention). Once the off-chip bandwidth is exceeded, however, CPI increases due to the limited service rate off-chip. The M term in Eq. 4 smooths out the curve for values of N near the saturation point, when queueing latency begins to lower CPI, but before throughput is completely dominated by off-chip bandwidth (Figure 1).

To confirm the model, consider the off-chip request rate, Λ, with CPI as derived by Eq. 8.

$$\Lambda = \frac{\Lambda_0}{\text{CPI}} \approx \begin{cases} \frac{\Lambda_0}{\text{CPI}_{\text{base}}} & \text{if } B > \frac{\Lambda_0}{\text{CPI}_{\text{base}}} \\ B & \text{otherwise} \end{cases}$$

$\Lambda \leq B$, as required, and the model is consistent.

Finally, the most important metric is the overall chip IPC. The off-chip request rate per core, $\Lambda_{\text{core},0} = \frac{\Lambda_0}{N}$, is the rate of memory requests per core in bytes per second. $\Lambda_{\text{core},0}$ depends only on the parameters of a single core, and is therefore independent of N. Using Eq. 8, our models shows how chip IPC scales the number of cores increases,

$$\text{IPC}_{\text{chip}} \approx \begin{cases} \frac{N}{\text{CPI}_{\text{base}}} & \text{if } N < \frac{B \cdot \text{CPI}_{\text{base}}}{\Lambda_{\text{core},0}} \\ \frac{B}{\Lambda_{\text{core},0}} & \text{otherwise} \end{cases} \quad (9)$$

IPC scales linearly with N until off-chip bandwidth has been exceeded. At this point, IPC is independent of N (in our model we assumed infinite queues at buffer points, so there is no degradation in performance). Thus, any N larger than $\frac{B \cdot \text{CPI}_{\text{base}}}{\Lambda_{\text{core},0}}$ does not contribute to performance.

We define the wasted area as the percentage of overall chip area that does not contribute to increased chip IPC. This is equivalent to,

$$\text{Waste} = \begin{cases} 0 & \text{if } N \leq \frac{B \cdot \text{CPI}_{\text{base}}}{\Lambda_{\text{core},0}} \\ \frac{N - \frac{B \cdot \text{CPI}_{\text{base}}}{\Lambda_{\text{core},0}}}{N} & \text{otherwise} \end{cases} \quad (10)$$

Intuitively, one expects that optimal processor configurations would have no waste. If there were waste, then some chip area would be better employed by increasing the size of caches to lower bandwidth requirements. Our results generally agree with this assessment (§IV-C).

III. Experimental Methodology

A. Cloud Applications

We believe that a driving application set for manycore processors is going to be cloud computing applications and data center aggregation. In this work we focus on this application set over more traditional parallel applications such as scientific applications. In the burgening field of cloud computing two of the most promising pieces of infrastructure for manycore chip designers are Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) applications.

Infrastructure as a Service providers supply the abstraction of a hosted whole machine virtual machine to end users. End users then craft their virtual machine image and the IaaS provider supplies a self-service interface to launch and control runing virtual machines. The IaaS provider consolidates many independant virtual machines onto a single multicore server. Example Public IaaS services include Amazon's EC2 [2], Rackspace's Cloud Server [3], and GoGrid [4]. Current uses of cloud computing infrastructure include webservers, databases, and general purpose compute serving. Due to the consolidation of many customer's workloads onto a machine, there are many independant workloads running thus providing the architecture with ample decoupled parallelism ideal for manycore architectures.

In Platform as a Service cloud applications, service providers provide a hosted development platform for developers to code against. Once an application is written to conform to the provider's API, the PaaS provider scales the developer's application relative to the application's demand. Example PaaS solutions include Google's App Engine [5], EngineYard, and Microsoft's Azure [6]. Currently these typically are used for developing dynamic web applications. The PaaS provider aggregates and distributes the many independant web connections across its hosted farm of multicore servers.

Both IaaS and PaaS cloud computing platforms exhibit large a number of independant parallel applications being hosted on current day multicore, and future manycore processors. Also, the current usage for IaaS and PaaS systems is integer codes. This is in sharp contrast to other parallel workloads such as scientific applications where floating point applications abound and data sharing between threads is common.

In order to model these cloud computing applications, we have used a workload of running independant SPEC Int 2000 applications on each core. Other workloads which have similar independant parallel computations are packet processing and the processing independent video/voice streams.

B. Evaluation Parameters and Base Architecture

When evaluating the best manycore configuration, there are many parameters and machine organizations which can be evaluated. We elected to model what we consider to be the canonical manycore processor. This canonical manycore

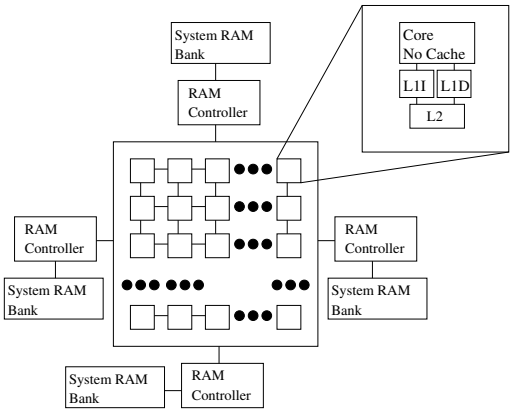


Fig. 2. An overview of the modeled system with. Enlargement of single core shown with included L1I, L1D, and unified L2 Cache.

processor is a homogeneous multiprocessor processor where the cores are arranged in a 2D grid connected by a mesh network. Each core contains a L1 instruction cache, an L1 data cache, and a unified L2 cache. The model assumes a single cycle per network hop and that the chip has four memory controllers which are each located at the center of the four edges. Figure 2 shows an overview of the modeled processor. This closely resembles the commercial Tilera Tile processor design [23].

Table II presents the parameters which are fixed across all of the calculations presented in this paper. These numbers were chosen to approximate current best-of-breed homogeneous manycore processors. The overall chip die area dedicated to processing was fixed at 200mm^2. This was chosen to approximate the economic die area which modern microprocessors use. Note that the overall die area does not change with process and that the listed die area is the die area without I/O. Processor frequency was chosen to model a 750MHz processor at 90nm with appropriate scaling trends. Area per core without any of the caches was chosen to be 1.5mm^2 in 90nm. This area was chosen to approximate a two to three wide issue microprocessor with in-core networks. This area, clock frequency, and performance are closely calibrated with the commercial, shipping Tile64 processor described in ISSCC [8] which is built in 90nm process technology and nominally runs at 750MHz. We chose to model a simple in-order processor such as the Tile64 instead of large cores such as the latest Intel Core offering becasue cloud computing providers are driven by efficient throughput oriented computation. Simple cores such as the Tile64 core afford better area and power efficiency when it comes to providing computational throughput per power and die area. The trend towards simpler cores in manycore architectures can be seen in the throughput oriented Sun Niagra 2 processor and the upcoming Intel Larrabee [18] processor. We chose a base, non-memory CPI to be 0.833 which is the average CPI for SpecINT 2000 on a simple in-order two way processor.

DRAM latency in absolute time is staying approximately constant across off-chip DRAM technologies such as DDR, DDR2, DDR3. This is because DRAM arrays are growing in size, counteracting the speed gains from process scaling. We chose the off-chip RAM bandwidth to approximate four RAM buses with 64-bit data-paths. At 90nm, the nominal off-chip RAM bandwidth is 17.88GB/s which is equivalent to four DDR2-533 RAM banks. At 32nm, the RAM Bandwidth is 53.69GB/s which is equivalent to four DDR3-1600 RAM banks. A linear bandwidth scaling assumption was used for the 65nm and 45nm nodes.

C. Evaluation Architecture

The fitness criteria for each modeled manycore configuration is the achieved IPC per core multiplied by the number of cores. This is the aggregate chip-wide IPC and signifies the overall computational bandwidth of applications assuming there is no shared data between the cores. This metric does not optimize for computational latency, but rather solely optimizes for throughput computing.

The methodology used to evaluate the best performing manycore cache configuration was to exhaustively evaluate our developed model across different cache configurations. The caches were evaluated over the following cache parameters:

Embedded DRAM was explored for usage in the L2 cache and the L1 Data cache. This is motivated by the fact that as process parameters shrink, SRAM gets slower, but DRAM is getting proportunately less slow as verified by the CACTI model. Also, the clock frequencies about which this design evaluates are modest enough to allow embedded DRAM to fit in a small number of core cycles making it feasible for a multi-cycle L2 cache.

The chip-wide IPC is computed for all the different benchmarks in SPEC Int 2000 individually. The arithmetic, geometric and harmonic means are also computed from the respective benchmark's chip-wide IPC. In addition to the chip IPC, the number of cores for each configuration along with the wasted area, as defined in Eq. 10 is computed.

The presented model is written in the Python programming language and takes approximately one hour to generate the results. 177,408 unique cache configurations are evaluated for each benchmark and process generation. The results are evaluated at the 90nm, 65nm, 45nm, and 32nm process nodes.

D. Data Source Integration

A key parameter to this study is the cache miss rates for the differing cache sizes and cache associativities. The cache miss rates for the SPEC Int 2000 Benchmark suite were used. This cache miss data was leveraged from the study done by Cantin and Hill [9], [10]. This study varied cache size from 1KB to 1MB by powers of two for both instruction, data, and unified caches. The complete benchmarks with full working sets were simulated on an Alpha based SimpleScalar processor model. In our study we model caches up to 8MB which is larger than the largest cache size (1MB) simulated in Cantin and Hill. In order to generate miss rates for the 2MB, 4MB, and 8MB cache sizes, a power curve fit was used to fit the last 7 points from the data set.

Process (nm)	90	65	45	32
Cycle Time (ns)	1.35	0.9	0.6	0.4
Frequency (GHz)	0.741	1.111	1.667	2.500
Non-Cache Core Size (mm^2)	1.5	0.75	0.375	0.1875
Total Die Area (mm^2)	200	200	200	200
DRAM and Controller Latency (ns)	100	100	100	100
RAM Bandwidth (GB/s)	17.88	29.82	41.75	53.69
Non-memory CPI	0.833	0.833	0.833	0.833

TABLE II
PHYSICAL PARAMETERS USED ACROSS PROCESS TECHNOLOGY.

L1 Instruction Cache Size (KB)	8, 16, 32, 64, 128, 256, 512
L1 Instruction Cache Associativity (sets)	Direct Mapped, 2, 4, 8
L1 Data Cache Size (KB)	8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192
L1 Data Cache Associativity (sets)	Direct Mapped, 2, 4, 8
L1 Data Cache Type	SRAM, DRAM
L2 Cache Size (KB)	0, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192
L2 Cache Associativity (sets)	4
L2 RAM Type	SRAM, DRAM

TABLE III
MEMORY HIERACHY OPTIONS USED IN STUDY.

In order to accurately evaluate cache sizes and cache timing, CACTI 5.3[22] from HP Labs was used. CACTI 5.3 has been updated over previous CACTI models to include scaling information from the latest ITRS roadmap for process technologies down to 32nm. CACTI was used to generate timing and area information for both high-speed SRAM and logic process DRAM. This cache information was gathered by script and turned into machine readable Python tables which fed the model.

E. Model Distribution

Good analytic models are key to understanding complex systems such as manycore grain size. Analytic models can provide the mathematical rigor and explanations for observed empirical trends that purely empirical studies cannot. In order to promote manycore grain size understanding, we are releasing our model, data sets, and source code in an open source manner to the architecture community. We hope that these models can be used as a guide for future manycore chip design.

IV. RESULTS AND ANALYSIS

We being the results section by presenting the chipwide IPC as a function of L1 and L2 cache sizes. Figure 3 plots the geometric mean of chip IPC against different cache sizes for SRAM and DRAM L2 caches. These numbers were gathered over the SPEC Int 2000 benchmark suite, as described in the previous section. The xy-plane shows L1 data cache size and L2 cache size. The graphs are plotted on log scale so that each point corresponds to a factor of two increase in cache size. The exception is the first L2 data point, which corresponds to no L2 cache. The largest caches appear in the left-most region of the graph, and the smallest on the right.

The graphs have been rotated so that the optimal configuration and points around it are clearly displayed. This puts the largest L2 configurations on the left edge "close to" the viewer, and the largest L1D on the left edge "away from" the viewer.

Graphs for each benchmark have the same general shape as the geometric mean. The SRAM plot (Fig. 3A) is a good example. Large caches yield poor performance (both edges along the left), as do small caches (the right-most region). The maximum usually occurs with a small L1D and medium-sized L2. The maximum is part of a ridge that occurs with L2 at the optimal size, indicating that this L2 size yields good performance for a variety of L1D sizes. There is a second ridge of generally lesser performance that occurs with an L1D of the similar size as the optimal L2. This ridge occurs because all L2's of smaller size than the L1D yield the same miss rate, and therefore similar chip IPC. (These configurations are a bit unrealistic, except for the point corresponding to no L2 cache.)

These graphs indicate a few results. DRAM L2 caches show significantly better peak performance, and the optimal DRAM configuration occurs with a much larger L2 than the optimal SRAM configuration. SRAM and DRAM show near identical performance for small L2s, but whereas the performance for DRAM continues to grow with large L2's, the performance of SRAM quickly deteriorates. Presumably this is due to the much greater area required for a large SRAM cache, reducing the number of cores.

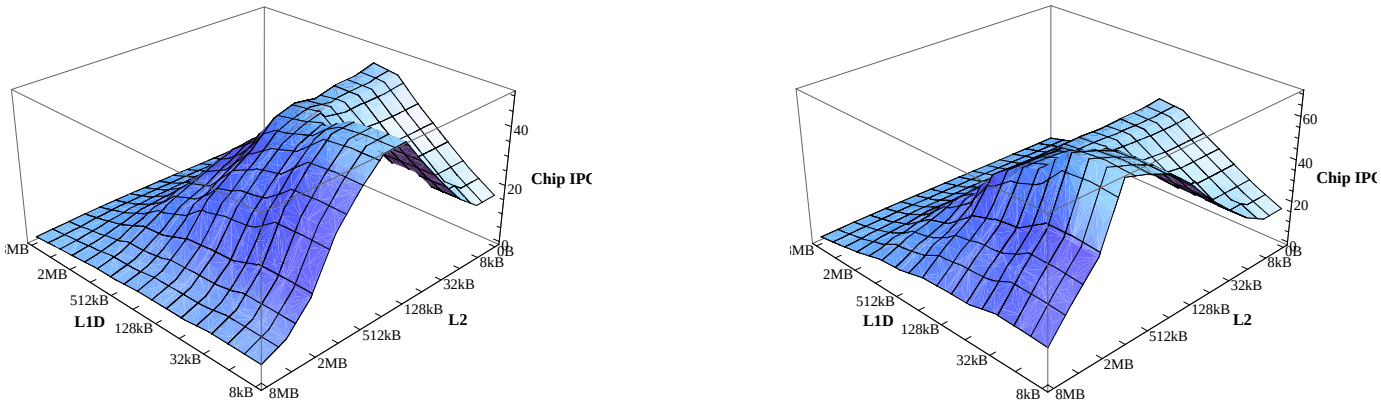


Fig. 3. Geometric Mean of Chip IPC for SRAM L1D at 32nm. (A) With SRAM L2. (B) With DRAM L2.

A. Fittest Configurations

In order to see scaling trends, we computed the chip IPC for all cache configurations and then gathered the optimal memory hierarchy configurations into Table IV. Table IV displays the cache configuration as a 5-tuple consisting of L1 instruction cache size, L1 data cache size, L2 cache size, number of cores, and overall chip IPC. The 5-tuple omits cache associativity, as associativity was found to be a second order effect. The rest of this section focuses on the geometric mean as labeled in the table as "gmean".

The first insight which can be gathered from this data is that as process technology progresses, the optimal cache size per core grows larger. This can be seen in both when the L2 cache is composed of SRAM and when it is composed of DRAM. For example, the optimal L2 cache size when composed of embedded DRAM grows from 128KB in the 90nm node up to 1MB at 32nm. This cache growth is due to the reduction of off-chip memory bandwidth per-core caused by exponential area scaling with only linear memory bandwidth scaling. Therefore instead of using increased area provided by process scaling for additional cores, the optimal configuration is to build larger caches to reduce per-core miss rate.

Second, it is surprising to see that the optimal configuration in 90nm with an SRAM L2 contains no L2 cache, but has 16KB L1I and 32KB L1D caches. This is largely due to the fact that decent sized caches can be built at the L1 level in 90nm and still fit in a single cycle. The model shows that it is not a good tradeoff to increase cache size at the expense of core count at 90nm in SRAM. Unfortunately, many times L2 caches are used to decouple a main processor pipeline from the rest of the cache system, but at larger feature sizes may not be the optimal design. At all process nodes, the area advantage gained from using DRAM as an L2 cache makes it worth building an L2 cache.

Another insight from this model is the factor by which L2 caches are larger than L1 caches (when the model predicts that building L2 caches is appropriate). The model finds that L2 caches should be at least 8x the size of an L1 cache to make good use of area in a multicore context.

We also found that as process technology advances, core count always increases. This is not necessarily intuitive, as an optimal design may use increased die area for larger caches, but our study finds that the large multiplicative factor of adding cores is difficult to offset by lower miss rates. An example of where core count did not grow appreciably between process nodes is the progression from 45nm to 32nm with a DRAM L2. In this example, the L2 cache grows considerably, from 256K to 1MB, but the number of cores only grows modestly from 151 to 158 cores.

Last, we found that associativity was not a first order effect with respect to the fittest cache configurations. Cache capacity was more important and for many of the optimal configurations, the set of top configurations all had the same cache size with slightly different associativities.

B. Optimal Configuration Disparity

We found that the optimal cache configuration within one process technology varied widely from benchmark to benchmark. This poses a large challenge for the multicore processor architect attempting to choose appropriate core count and cache configurations.

Figure 4 shows overall chip IPC (z-axis) plotted against L1D and L2 cache sizes (xy-axis) for four benchmarks, gzip, gcc, mcf, and gap at the 32nm node with DRAM L2 caches. The first thing to note is that the highest fitness configuration is in wildly different locations on these plots. This result is exaggerated even more by the fact that the xy-axis for these plots is on a logarithmic scale.

Examining the plot for gzip (A), we find that the optimal configuration lies center of the front right face. This corresponds to a small (8KB in this case) L1D cache and a 256KB L2 cache. But when looking at gcc (B), the peak moves over to favor larger L2 caches sizes (1MB).

The shape of the fitness function is also not uniform across benchmarks as shown by the mcf (C) and gap (D) outliers. mcf shows that many locations have almost equal fitness along the top of two pronounced intersecting ridges. A large plateau is

		90 nm	65 nm	45 nm	32 nm
SRAM	gmean	16K, 32K, 0K, 70, 18.4	8K, 8K, 128K, 75, 27.4	32K, 8K, 128K, 139, 46.1	64K, 16K, 128K, 222, 50.6
	amean	16K, 32K, 0K, 70, 20.9	8K, 8K, 128K, 75, 35.9	16K, 8K, 128K, 146, 64.9	64K, 16K, 128K, 222, 83.6
	hmean	16K, 32K, 0K, 70, 14.1	64K, 64K, 0K, 78, 18.4	32K, 8K, 128K, 139, 23.1	64K, 8K, 256K, 152, 22.0
	164.gzip	8K, 128K, 0K, 48, 36.3	8K, 128K, 0K, 98, 70.5	8K, 128K, 0K, 205, 134.2	8K, 256K, 0K, 236, 192.5
	175.vpr	8K, 64K, 0K, 64, 18.0	8K, 128K, 0K, 98, 25.0	16K, 256K, 0K, 130, 29.9	16K, 16K, 512K, 116, 29.4
	176.gcc	16K, 32K, 0K, 70, 14.9	64K, 32K, 0K, 87, 20.4	32K, 256K, 0K, 121, 28.1	64K, 512K, 0K, 118, 32.3
	181.mcf	8K, 64K, 0K, 64, 3.1	8K, 128K, 0K, 98, 3.7	16K, 256K, 0K, 130, 3.6	16K, 512K, 0K, 132, 3.2
	186.crafty	64K, 64K, 0K, 38, 28.4	64K, 64K, 0K, 78, 55.6	64K, 64K, 0K, 160, 138.7	64K, 64K, 0K, 312, 193.3
	197.parser	16K, 32K, 0K, 70, 25.4	8K, 128K, 0K, 98, 35.3	16K, 128K, 0K, 187, 64.4	8K, 256K, 0K, 227, 71.5
	252.eon	8K, 8K, 64K, 44, 42.2	8K, 8K, 64K, 86, 81.9	8K, 8K, 64K, 205, 193.9	8K, 8K, 64K, 407, 378.9
	253.perlbmk	8K, 8K, 64K, 44, 31.4	8K, 8K, 128K, 77, 58.0	8K, 8K, 128K, 160, 107.5	8K, 8K, 128K, 318, 130.6
	254.gap	32K, 8K, 0K, 61, 31.6	64K, 8K, 0K, 98, 52.7	32K, 64K, 0K, 215, 94.7	64K, 32K, 0K, 396, 87.8
	255.vortex	64K, 64K, 0K, 38, 22.6	8K, 8K, 128K, 77, 39.0	64K, 64K, 0K, 160, 65.2	64K, 64K, 0K, 361, 79.2
	256.bzip2	8K, 16K, 0K, 85, 38.5	8K, 32K, 0K, 146, 56.6	8K, 128K, 0K, 205, 73.2	8K, 256K, 0K, 235, 77.0
	300.twolf	16K, 32K, 0K, 70, 14.3	64K, 128K, 0K, 68, 17.0	32K, 256K, 0K, 120, 23.9	64K, 16K, 1024K, 67, 26.9
		90 nm	65 nm	45 nm	32 nm
DRAM	gmean	16K, 8K, 128K, 53, 23.3	8K, 8K, 128K, 101, 31.9	32K, 16K, 256K, 151, 55.9	64K, 16K, 1024K, 158, 70.4
	amean	16K, 8K, 128K, 53, 28.2	8K, 8K, 128K, 101, 43.2	16K, 16K, 256K, 160, 79.1	16K, 16K, 256K, 370, 119.0
	hmean	16K, 8K, 128K, 53, 15.9	32K, 8K, 128K, 78, 19.4	32K, 16K, 256K, 151, 25.6	64K, 16K, 1024K, 158, 29.4
	164.gzip	8K, 8K, 128K, 56, 45.6	8K, 8K, 256K, 85, 78.9	8K, 16K, 256K, 172, 159.8	8K, 8K, 256K, 391, 343.0
	175.vpr	8K, 8K, 128K, 56, 19.3	8K, 8K, 256K, 87, 27.2	16K, 16K, 512K, 102, 32.5	16K, 16K, 1024K, 193, 46.2
	176.gcc	16K, 32K, 0K, 70, 14.9	8K, 8K, 1024K, 40, 21.7	32K, 16K, 1024K, 72, 37.2	64K, 16K, 1024K, 158, 64.6
	181.mcf	8K, 8K, 128K, 58, 3.1	8K, 128K, 0K, 98, 3.7	16K, 8K, 256K, 170, 3.6	16K, 8K, 1024K, 195, 3.6
	186.crafty	16K, 8K, 128K, 53, 35.5	8K, 8K, 128K, 104, 58.3	64K, 64K, 0K, 160, 138.7	16K, 16K, 256K, 370, 247.8
	197.parser	16K, 8K, 128K, 53, 26.6	64K, 8K, 128K, 69, 38.1	16K, 8K, 256K, 162, 78.9	8K, 16K, 1024K, 191, 114.7
	252.eon	16K, 8K, 64K, 56, 53.2	8K, 8K, 64K, 116, 99.7	16K, 16K, 64K, 207, 202.0	16K, 16K, 64K, 419, 401.7
	253.perlbmk	8K, 8K, 128K, 56, 43.3	8K, 8K, 128K, 104, 73.7	8K, 8K, 128K, 206, 128.5	8K, 8K, 256K, 401, 185.7
	254.gap	32K, 8K, 0K, 61, 31.6	64K, 8K, 0K, 98, 52.7	32K, 64K, 0K, 215, 94.7	64K, 32K, 0K, 396, 87.8
	255.vortex	8K, 8K, 128K, 56, 30.8	8K, 8K, 256K, 85, 49.4	16K, 8K, 256K, 169, 82.4	16K, 16K, 1024K, 193, 108.9
	256.bzip2	8K, 8K, 16K, 78, 39.4	8K, 8K, 64K, 113, 58.6	8K, 8K, 256K, 176, 85.6	8K, 8K, 1024K, 199, 124.7
	300.twolf	16K, 8K, 128K, 54, 14.8	64K, 8K, 1024K, 34, 18.3	32K, 16K, 1024K, 72, 35.1	64K, 16K, 1024K, 158, 55.4

TABLE IV

OPTIMAL CACHE CONFIGURATIONS FOR SPEC INT 2000 BENCHMARKS. ENTRY CONSISTS (L1I CACHE SIZE, L1D CACHE SIZE, L2 CACHE SIZE, NUMBER OF CORES (N), CHIP IPC (RELATIVE FITNESS))

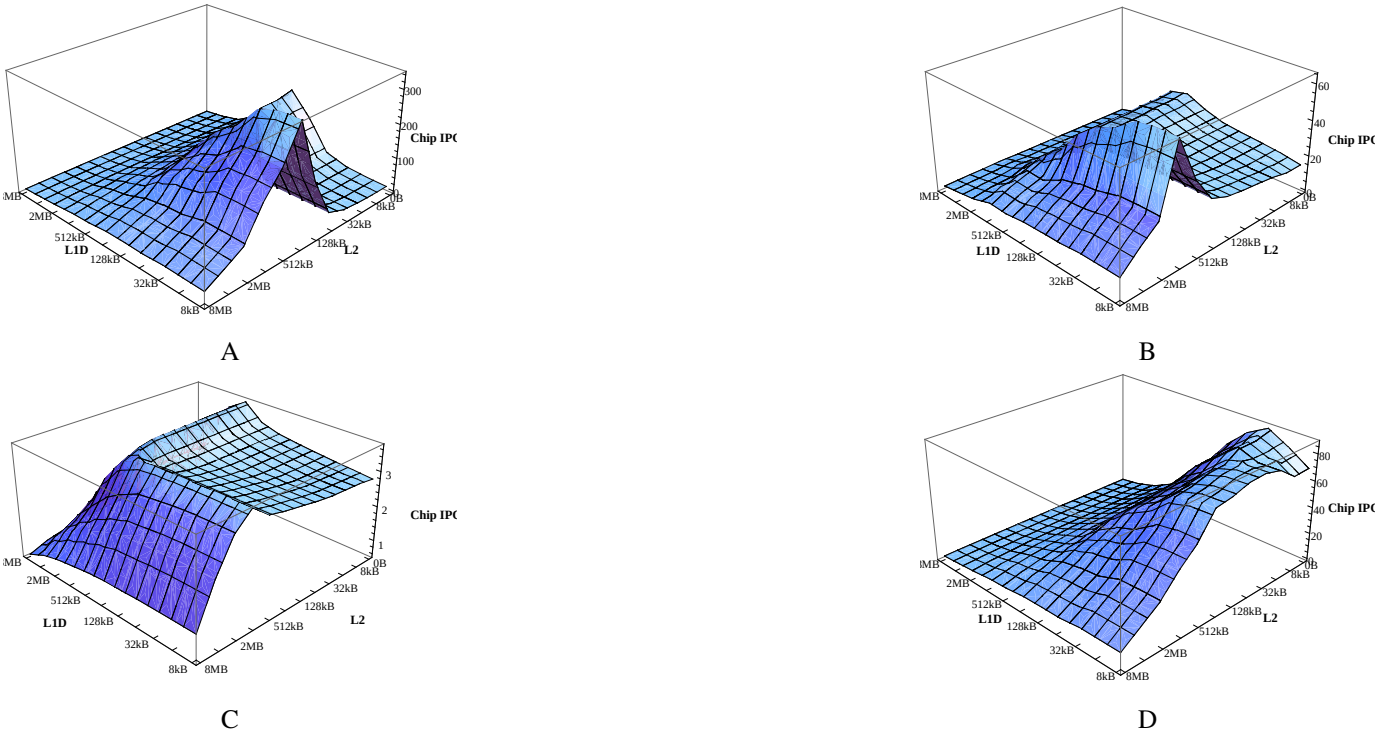


Fig. 4. Chip IPC for various benchmarks with DRAM L2 and SRAM L1D at 32nm: (A) gzip, (B) gcc, (C) mcf, (D) gap.

formed on the right corner of the mcf plot, which is interesting as it shows that mcf is not very sensitive to cache size in that region. This is likely due to the overall poor cache performance which mcf exhibits. Last, gap has its fitness graph skewed toward small L1 and L2 caches and is the only benchmark which has its optimal configuration with no L2 cache for any process node.

C. Wasted Area

Earlier in this paper we defined "waste" as the percentage of chip area which does not meaningfully contribute to performance. Specifically, this area is consumed by cores that produce excess memory requests after off-chip bandwidth has been exceeded. We claimed that optimal configurations would have minimal waste — the excess area could instead be used to build larger caches for the remaining cores, increasing chip throughput.

Figures 5, 6, 8, and 7 demonstrate this effect. As before, these plots have L1 data cache size and L2 size on the xy-plane. In graph A we plot chip IPC, and in graph B we plot waste. It's clear that chip IPC and waste follow a somewhat inverse relationship; if waste is high, then chip IPC is low. Optimal configurations occur on the transition from high waste to low waste. This is true even in extreme outliers, such as MCF (Fig. 6), which has very low performance; and GAP (Fig. 7), which recommends no L2 cache at all.

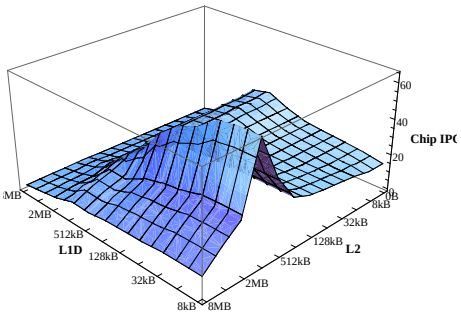
It is obvious that high waste should result in low chip IPC. High waste occurs with small cache configurations such that many cores fit on a single chip, but there are a great many misses to main memory. The large number of misses to memory means that it takes few cores to reach the off-chip bandwidth limit. Once off-chip bandwidth is exceeded, the large number of cores no longer contribute to performance (the definition of waste), and chip IPC is equivalent to a chip with few cores and high miss rates, giving low IPC.

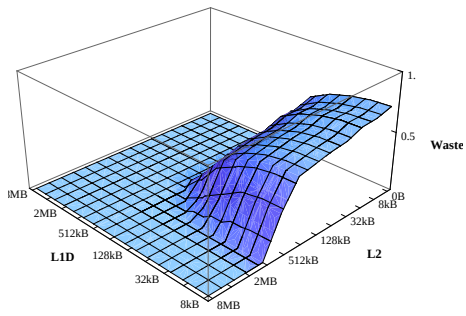
Similar reasoning explains why configurations with very large caches do not perform well. Performance is maximized when the maximal number of cores fit on chip without exceeding off-chip bandwidth. The optimal cache configuration will be the smallest that achieves this. Increasing the cache size beyond this point reduces the number of cores dramatically, so although larger caches might result in fewer misses and higher IPC per core, the reduction in number of cores is more significant.

This explains why the configurations in Table IV all show surprisingly low IPC per core. For example, the optimal configurations by geometric mean at 32nm had per-core IPC of 0.23 and 0.45 for SRAM and DRAM, respectively. Although configurations with larger caches achieve higher per-core IPC, they result in lower chip IPC (Figure 3).

Note that the individual benchmarks (Figs. 5, 6, 8, 7) have sharp peaks in chip IPC at precisely the point when waste drops. This situation changes when we consider mean performance (Figs. 9, 10). The same general relationship holds as before, but now the waste drops off gradually. This corresponds with a less pronounced peak in chip IPC, and means that the optimal configuration still has some waste.

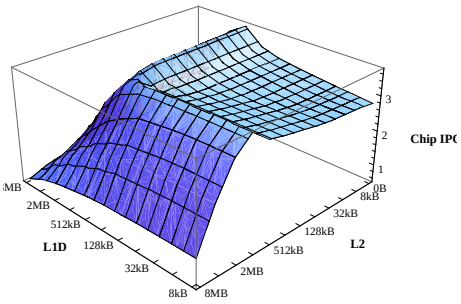
This is sensible, as we are now optimizing over a variety of applications. Each has different memory requirements, so the optimal mean configuration will certainly be suboptimal

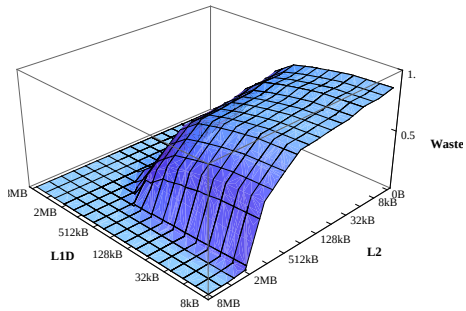




A B

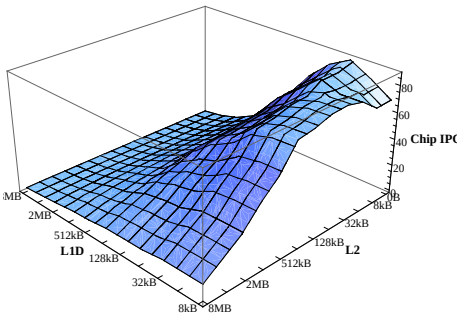
Fig. 5. Results for GCC with SRAM L1D and DRAM L2. (A) Chip IPC. (B) Wasted area.

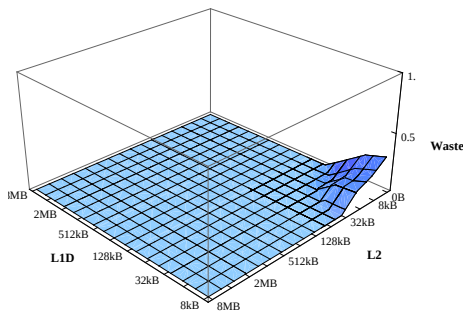




A B

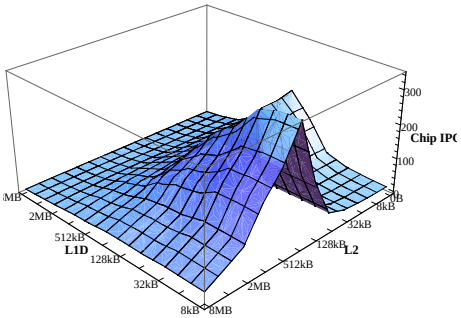
Fig. 6. Results for MCF with SRAM L1D and DRAM L2. (A) Chip IPC. (B) Wasted area.

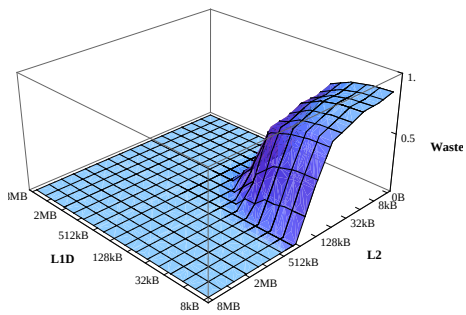




A B

Fig. 7. Results for GAP with SRAM L1D and DRAM L2. (A) Chip IPC. (B) Wasted area.





A B

Fig. 8. Results for GZIP with SRAM L1D and DRAM L2. (A) Chip IPC. (B) Wasted area.

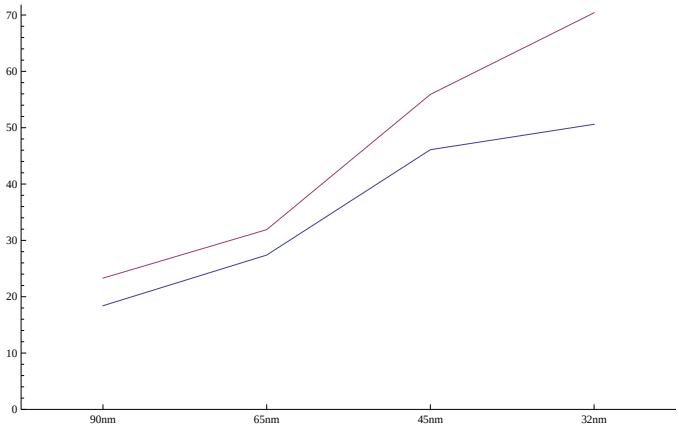


Fig. 11. Chip IPC for optimal DRAM configurations (red) vs. optimal SRAM configurations (blue) at each process node.

for individual benchmarks. For some of the benchmarks, we will not use the full off-chip bandwidth capacity. For others, we will slightly exceed off-chip bandwidth. This gives rise to modest waste in the optimal configuration (10-25% for SRAM and 5-15% for DRAM).

D. Effect of Embedded DRAM on Performance

Figure 11 plots peak chip IPC for DRAM and SRAM L2 caches over all process nodes, measured by geometric mean. It was found that DRAM L2 caches gave superior chip IPC than SRAM in all cases, and the difference was larger at more advanced process nodes. Consulting Table IV, DRAM performs better over all individual benchmarks as well.

The tradeoff between SRAM and DRAM is between cache area (mm^2) and access time. SRAM has lower access time but takes roughly four times as much area for the same cache size. Our model indicates that the area savings of DRAM are more significant. Furthermore, DRAM configurations did not always exhibit more cores than SRAM configurations, as one might expect from the area savings, but rather favored larger caches.

Also note that even at 32nm, the optimal DRAM configuration does not indicate all of system RAM should be integrated on chip. The total cache capacity is 170MB when all of the cache area is used for embedded DRAM, whereas main memory capacity will certainly exceed 10GB. This tradeoff likely will change as more RAM fits on-chip and per-core off-chip memory bandwidth becomes more limiting as predicted in future process nodes

V. Related Work

There have been several previous studies that have examined the problem of partitioning finite resources in processor architectures.

Multi-level cache hierarchies have been studied in uniprocessors several researchers including Jouppi et al. [15]. This study explores the space of two-level hierarchies by analyzing the area cost, access time, and combined miss rates of various combinations of L1 and L2 caches. Our work builds on this by: considering embedded DRAM caches; using more advanced, contemporary cache models; extending it to manycore processors where there is competition for DRAM bandwidth; and considering a throughput oriented application mix.

SimpleFit [16] is a purely analytical framework used to explore the design space of multicore processors. The authors derive simple models for the die area consumed by various processor components (computation, on-chip memory, on-chip networks, and off-chip I/O) and the computational, memory, and communication needs of various applications. They use a performance function to estimate the run time of a particular application model on a specific processor model. They then attempt to optimize this function by varying the processor component models given a fixed total area budget.

Whereas SimpleFit examines many different tradeoffs at a very high level, this work focuses on the memory system and explores it in much greater detail. The SimpleFit authors briefly examine the question of using embedded DRAM for local memories and find (as we do) that it results in significantly better overall performance. By using CACTI models and actual measured cache miss rates, our analysis is much more detailed and includes a greater number of memory configurations. It is interesting to note that Moritz et al. find that the optimal processor configuration contains a very large number of cores with small memories and large on-chip networks while our analysis favors fewer cores with larger memories. This makes sense given their focus on parallel applications with high degrees of communication and relatively small working sets rather than total throughput of larger independent applications.

Thoziyoor et al. [21] introduced CACTI-D and used it to perform an evaluation of different implementation technologies for L3 caches. The study assumed a fixed base architecture consisting of eight processing cores with SRAM-based L1 and L2 caches and added a stacked die to provide a large shared L3 cache. L3 caches based on SRAM, embedded (logic-process) DRAM, and commodity DRAM were compared. As with our study, the authors found that the DRAM-based caches outperformed the SRAM-based cache. However, they did not consider alternate configurations of the L1 and L2 caches as we do, nor do they consider trading memory for additional processing cores to find an optimal balance between memory and computation.

Huh et al. [14] explores the design space for future chip multiprocessors. Huh differs from this work in that it is only a simulation based study which differs from this work which contains a model and hence can be used to draw further conclusions beyond what is presented in this paper. We also search a wider search space. Huh also does not explore embedded DRAM. The core that they evaluate is twice the size of our evaluated core size, and finally Huh subsets SPEC Int 2000 while our study utilizes the complete benchmark.

Rogers et al. [17] also studies bandwidth constrained multicore designs. Rogers explores a completely analytic model, while our work explores an analytic model and a simulation fed model. Rogers does explore a wide variety of solutions to the DRAM scaling problem while our work more fully inves-

A B

Fig. 9. Results for geometric mean over SPECINT2000 benchmarks with SRAM L1D and SRAM L2. (A) Geometric mean of chip IPC. (B) Mean wasted area.

A B

Fig. 10. Results for geometric mean over SPECINT2000 benchmarks with SRAM L1D and DRAM L2. (A) Geometric mean of chip IPC. (B) Mean wasted area.

tigates DRAM vs SRAM tradeoffs in manycore processors.

VI. Future Work

In the future, we would like to extend this model to include an L3 cache. Extending our model to include a non-shared L3 cache is a simple extension. Extending the model to include shared L3 caches or distributed L3 caches is complicated by the fact that inter-core sharing must be modeled. We believe that the sharing factor would vary widely across different workloads. There is even the possibility of code sharing in throughput oriented applications.

One limitation in our model which we would like to improve is that the model currently uses a fixed IPC for non-memory instructions for all benchmarks. We would like to model the non-memory IPC on a per benchmark basis. Unfortunately this parameter is very dependent on micro-architecture specifics. We do not think that modifying this parameter on a per benchmark basis would qualitatively change our findings, but we would like greater model fidelity.

To gain greater model fidelity, we would like to add the ability to model out-of-order memory systems. While throughput oriented computations do favor area and power efficient architectures, we are interested in whether adding an out-of-order memory system to in-order processors would better hide latency of the the main memory system. Ultimately, adding an out-of-order memory system does not change the needed off-chip memory bandwidth, but we are still interested in modeling the effect of this architectural feature.

Last, we would like to extend this grain size study to not only cache sizes. Primarily, we would like to vary the non-cache core area along with non-memory IPC to see whether the optimal multicore grain size favors wider-larger cores or narrower-smaller execution resources.

VII. Conclusion

In conclusion, we believe that poor scaling of off-chip memory bandwidth when compared to on-chip silicon resources will continue to motivate the exploration of multicore cache hierarchies. In this work we have constructed a framework to study cache size versus core count which is the primary tradeoff in multicore design. This framework was utilized to determine optimal core count and cache configurations for multiple future process technologies. This tool will serve as a guide for multicore chip architects by enabling the quick exploration of a large space of possible configurations.

References

[1] Intel Microarchitecure (Nehalem), 2008. http://www.intel.com/technology/architecture-silicon/next-gen/index.htm.

[2] Amazon elastic compute cloud (amazon ec2), 2009. http://aws.amazon.com/ec2/.

[3] Cloud hosting products - rackspace, 2009. http://www.rackspacecloud.com/cloud_hosting_products.

[4] Gogrid cloud hosting, 2009. http://www.gogrid.com/.

[5] Google app engine - google code, 2009. http://code.google.com/appengine/.

[6] Windows azure platform, 2009. http://www.microsoft.com/azure/.

[7] A. Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, 2:398–412, 1991.

[8] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, and M. Reif. TILE64 Processor: A 64-Core SoC with Mesh Interconnect. In *International Solid-State Circuits Conference*, 2008.

[9] J. F. Cantin and M. D. Hill. Cache performance for selected SPEC CPU2000 benchmarks. *SIGARCH Comput. Archit. News*, 29(4):13–18, 2001.

[10] J. F. Cantin and M. D. Hill. Cache performance for spec cpu2000 benchmarks: Version 3.0, 2003. http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data.

[11] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar. An integrated quad-core Opteron processor. In *Proc of the IEEE International Solid-State Circuits Conference (ISSCC)*, pages 102–103, Feb 2007.

[12] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki. Synergistic processing in Cell's multicore architecture. *IEEE Micro*, 26(2):10–24, March-April 2006.

[13] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, San Francisco, CA, third edition, 2003.

[14] J. Huh, D. Burger, and S. W. Keckler. Exploring the design space of future cmps. *Parallel Architectures and Compilation Techniques, International Conference on*, 2001.

[15] N. P. Jouppi and S. J. E. Wilton. Tradeoffs in two-level on-chip caching. In *ISCA '94: Proceedings of the 21st International Symposium on Computer Architecture*, pages 34–45, Apr 1994.

[16] C. A. Moritz, D. Yeung, and A. Agarwal. Simplefit: A framework for analyzing design trade-offs in Raw architectures. *IEEE Trans. Parallel Distrib. Syst.*, 12(7):730–742, Jul 2001.

[17] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the bandwidth wall: challenges in and avenues for cmp scaling. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 371–382, 2009.

[18] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: A many-core x86 architecture for visual computing. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–15, New York, NY, USA, 2008. ACM.

[19] B. Stackhouse, B. Cherkauer, M. Gowan, P. Gronowski, and C. Lyles. A 65nm 2-billion-transistor quad-core Itanium processor. In *Proc of the IEEE International Solid-State Circuits Conference (ISSCC)*, pages 92–598, Feb 2008.

[20] TMS320C6474 multicore digital signal processor, Oct. 2008. http://www.ti.com/lit/gpn/tms320c6474.

[21] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, pages 51–62, Washington, DC, USA, 2008. IEEE Computer Society.

[22] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. Cacti 5.1. Technical Report HPL-2008-20, Hewlett-Packard Labs, 2008.

[23] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, Sept. 2007.