Cache Capacity and Memory Bandwidth Scaling Limits of Highly Threaded Processors

Jeff Stuecheli[12] Lizy Kurian John[1]

[1]Department of Electrical and Computer Engineering, University of Texas at Austin

[2]IBM Corporation Austin, TX

Abstract

As computer architects search for greater system performance from increasing transistor budgets, diminished returns from feature-size scaling have made performance improvements for single-threaded programs more difficult. As a result, designers have proposed various microarchitectural techniques for parallel execution of multiple program contexts within a single packaged processor. However, as additional program contexts share the limited resources of cache capacity and external memory bandwidth, throughput becomes constrained. We investigate the inflection point at which additional thread contexts begin to decrease throughput and power efficiency. The SPEC CPU 2006 floating point and integer workloads are evaluated across a range of chip thread counts. Scaling projections based on the results of these microarchitectural simulation are developed and compared against existing hardware systems. The results are then used to propose hardware/software systems that optimize the system throughput and power efficiency in systems with a large number of simultaneously executing program contexts.

1.0 Introduction

Computer system development is driven by increases in system capability. New systems must have greater capability than previous designs to warrant development expense. Over the past decades, much of this increased capability was provided through greater system execution performance. Smaller and faster transistors have enabled greater performance through increases in single-threaded execution speed.

Recently however, computer designers have been unable to translate transistor technology improvements into single instruction stream performance gains. This is due to an increase in relative wire delay, single-thread scalability issues, and increased power constraints due to leakage current in smaller transistors.

Due to difficulties in increasing single instruction stream performance, computer designers have proposed additional threads of execution to increase system throughput. For a given amount of silicon, additional execution threads have been implemented by a combination of multiple execution cores on one die, known as chip multi-processing (CMP), and complex execution cores that manage more than one thread of execution, including simultaneous multi-threading (SMT) and hardware multi-threading (HMT)[13].

The increased core count on a die, or packaged CPU, has put pressure on two key resources: cache capacity and memory bandwidth. As the execution bandwidth of a processor module increases due to higher core count, memory bandwidth climbs at a slower rate [19]. In addition, the cache capacity available per thread of execution decreases as more threads are packed into the same limited silicon.

The goal of this research is to investigate how the effects of the constrained design resources compound as thread counts increases. In general, we investigate the inflection point where additional program contexts cause decreased system throughput. Analysis also shows variation of this point across a mix of workloads in the rate versions SPEC CPU 2006 suite. Due to the workload dependant nature of the system response, we investigate system-level optimization mechanisms that mitigate the performance loss by detecting throughput limits. The optimization is also considered relative to power consumption.

1.1 Outline

This paper is structured as follows:

- **System constraints:** We investigate current and future system cache and memory capabilities in Section 2.0.
- **Simulation/Analytical analysis:** A combination of simulation and analysis is used to estimate the effects on cache and memory constraints as additional program threads are added. This is described in Section 3.0.
- **System measurement:** Measurements are taken on existing CMP/SMT systems. These will be compared with the analytical findings in an effort to validate the premise in Section 4.0.
- **Benchmark Analysis:** The various workload components of the SPEC CPU 2006 [22] suite are investigated and classified based on observed characteristics in Section 5.0.
- **System optimization:** In Section 6.0, we focus on system optimization techniques that can be enabled by hardware/software features.

Processor	Number Cores	Number Threads	Last Level Cache	Last level Cache sharing group	Cache per Thread	Process
Sun UltraSPARC T2	8	64	4 MB	64 threads	64kB	65nm
Sun UltraSPARC T1	8	32	3 MB	32 threads	96kB	90nm
Intel Xeon	4	4	8 MB	2 cores	2MB	45nm
Intel Itanium2 9000	2	4	24 MB	2 threads	12MB	65nm
AMD Barcelona	4	4	2-6 MB	4 cores	512kB-1536kB	65nm
IBM POWER6	2	4	32 MB	4 threads	8MB	65nm

TABLE 1. Contemporary system cache size and thread count.[10][11][12][13][15][16]

2.0 Background and System Constraints

Socket capability is limited by physical constraints. The socket constraints include the physical package and pin count that are available to the chip. The limitations include the number of communication pins, current delivery, heat removal, and chip die size. These constraints can be relaxed beyond typical limits by trading off cost. An example is multiple fabricated chips packaged into one socket using 2D packaging, such as multi-chip module (MCM) technologies [18][12], or with emerging 3D stacked die techniques [3][4][7]. These techniques comprise a means of *generation stretching* because the capabilities of future mainstream technologies are realized with some additional cost.

This study focuses on two specific socket-based constraints. The available silicon area is the first constraint. Specifically, as more central processing units are fabricated on a die, the available area for a cache is constant or decreasing. The cache area can stay constant with additional CPUs by using two primary methods. First, the yield of a silicon die can be improved using logic rather than cache area by the addition of built-in redundancy in the cache arrays. An example of this design trade-off is seen in the dual-core Itanium2 processor in which the yield of the very large die area is made possible by redundant cache arrays [14]. Second, as logic pushes cache out of the processor chip, constant cache area can be reestablished using multiple-chip modules [18] and stacked chips [12].

Table 1 contains the cache sizes and thread counts found in recent commercial designs. We use these sizes as a guideline for analysis. Two very different design paradigms are visible here. The IBM POWER6 and Intel Itanium2 have extremely large caches compared to the other designs. At the other end of the spectrum, the Sun T1 and T2 designs have extremely high thread counts and very modest cache sizes. The Sun T1 and T2 machines hide the higher cache miss rates behind many thread contexts and buffered bandwidth to memory. For highly-threaded workloads this proves very effective.

The second primary socket constraint is pin bandwidth. In these studies, we focus specifically on bandwidth to main memory. Various technologies provide higher bandwidth for a given number of chip I/O at marginal additional cost. One prominent method is to add a buffering chip between the DRAM chips and the processor chip. This off-chip memory buffer converts the relatively slow bi-directional DRAM interface into multiple high speed uni-directional interfaces. The Fully Buffered Dual Inline Memory Module (FB-DIMM) standard uses differential pair interfaces to run at six times the DRAM frequency [1]. The FB-DIMMs increase bandwidth in two ways. First, each pin drives three times the bandwidth of a traditional double-data rate (DDR) pin. In addition, the FB-DIMM standard allows for daisy-chaining of DRAM devices to achieve higher pin utilization compared to raw DDR interfaces that incur penalties dues to read/write bus turnaround and switching between multiple DRAM devices that share a common bus. Another high bandwidth memory technology uses a buffering chip developed by IBM called synchronous memory interface (SMI) [18]. In this interface, a single-ended Elastic I/O [8] is used to drive each pin at twice the DRAM frequency. Each pin carries two times the traditional DDR pin bandwidth. In addition, the SMI chip has multiple DRAM ports that feed the high bandwidth uni-directional interface back to the CPU chip, achieving higher pin utilizations at the processor chip much like FB-DIMM. One disadvantage of the uni-directional interface compared to the standard DDR interface is the fixed read and write bandwidth ratios. In DDR, all of the pin bandwidth can be dedicated to one type of traffic, whereas the uni-directional pin forces a specific mix of inbound and outbound bandwidths. These interfaces are built around a 2:1 mix of read and write traffic, which is an approximation of the traffic ratio in many commercial and technical workloads.

Ganesh *et al.* [2] provide the typical FB-DIMM bandwidth characteristics relative to traditional *direct-connect* DDR memory. The following table shows the sustained socket bandwidth for contemporary processor chips running the Stream Triad benchmark [21]. The highest sustained result is achieved with the IBM SMI design. Currently, no stream result has been published for an FB-DIMM based system, but Intel Xeon systems are available with FB-DIMM chipsets. In addition, the recently announced FB-DIMM based UltraSparc T2 system claims bandwidth of 60 GB/sec. [11].

System	Socket stream triad (GB/sec)
IBM POWER5+	12.6
AMD Quad FX-72	7.6
HP AlphaServer GS1280-1300	7.2

TABLE 2. Stream Triad benchmark results[21]

System	Socket stream triad (GB/sec)
Intel Core2	5.8
HP Integrity rx6600 (Itanium2)	5.6
Fujitsu/Sun_SPARC_Enterprise_M9000	3.5
Sun SPARC64 VI	4.5

TABLE 2. Stream Triad benchmark results[21]

Beyond these two aspects, power consumption must also be addressed. Power is not specifically modeled in the simulation, but the effect can certainly drive system level controls and optimization. Power consumption controls in CMP, HMT, and SMT designs are useful for optimizing power/performance tradeoffs. P.A.Semi has indicated per core voltage and frequency control capability in its PA6T design [23]. Per core voltage control of an aggressive CMP design would be enhanced by integration of the Voltage Regulator Module (VRM) using 3D packaging [5]. Without 3D VRM integration, the cost of delivering many voltage domains would make it infeasible. Power dissipation control for HMT and SMT systems are not feasible through voltage rail control since the logic across threads is tightly coupled. For these cases, footer transistor devices can be used to shut off leakage from unused devices that share voltage rails with active logic [9]. An example is varying-off architects state of unused threads in a SMT processor.

3.0 Simulation Based Analysis

The goal of this work is to assess how many threads a socket can support in realistic Microarchitecture across a mix of workloads. There is significant variation in the throughput inflection point across different workloads. Analysis of the workloads in conjunction with their inflection points exposes system optimization possibilities not exploited by current system environments.

The experimental methodology centers around the cache simulations of trace samples of the SPEC CPU2006 workloads. Each benchmark trace is simulated with a range of cache sizes between 4 MB and 32 KB. The cache simulations produce miss rates per instruction for the last level cache, which are then extrapolated to memory bandwidth requirements. The projected memory bandwidth is then compared with available socket memory bandwidth.

3.0.1 Simulation Methodology

We use trace samples obtained from a full execution of the SPEC CPU 2006 binaries using the IBM Aria methodology [30]. Performance counter and simulation data was collected and analyzed to identify sample points that represent the overall performance of the full binary execution. Representative traces of 100M instructions per workload were obtained dynamically by instrumenting the binaries and collecting instruction traces at the identified sample points [27][28][29][30]. These traces were found from experiments to correctly predict the performance of the full binary in hardware when executed in a highly-detailed and validated POWER5 M1 simulator [27][28].

A cache simulator was used to generate a cache miss rate for the collected traces over various cache sizes. The following cache sizes were simulated:

4 MB, 2 MB, 1 MB, 512k, 256k, 128k, 64k, 32k.

These sizes are chosen to reflect a maximum 4 MB last level cache per thread with a doubling of the thread count for each half-sized cache. This enables a projection range of 1 to 128 threads sharing a 4 MB last level cache and memory controller. The simulated data can also be used for larger shared caches. For example, a 16 MB shared cache would correspond to a system with 4 to 512 threads. Our method is simplistic in that it assumes the threads do not share any data and each thread gets an even fraction of the last level cache. This assumption serves to isolate the effects of capacity reduction. More destructive interference between threads is not modeled. The "rate" version of SPEC CPU is non-sharing and homogeneous, thus it can be accurately projected given these modeling assumptions.

3.0.2 Scaling Projections

Each of the cache simulation results is used to extrapolate the memory bandwidth requirements for a thread.

The cache miss rate is multiplied by the cache line size to generate the bytes read from memory for each instruction:

$$\frac{Bytes}{Instruction} = MissRate_{thread} \times LineSize$$

The bandwidth requirements for each thread then becomes the product of the bytes/instruction, CPU frequency, and the instructions/cycle executed:

$$BW_{thread} = \frac{Bytes}{Instruction} \times Frequency \times IPC$$

The total bandwidth required is then calculated as the sum of the bandwidth for each thread in the socket:

$$BW_{required} = \sum BW_{thread}$$

If the required socket bandwidth exceeds the available memory bandwidth, the throughput is degraded:

$$Throughput = min\left(1, \frac{BWsocket}{BW_{required}}\right) \times NumThreads$$

For the baseline analysis, a line size of 64 bytes, 2GHz clock rate, 40 GB/sec of memory bandwidth, and an IPC of 1 are assumed.

3.1 Simulation Results

Scaling factors were generated for each workload in the SPEC CPU2006 integer and floating point benchmarks over the range of 1 to 128 threads. These are shown in Table 3 and Table 4 on page 4.

The various workloads can be classified into three phases or *regions* with respect to the scaling of additional execution threads, numbered as follows:

1. **Increasing**: The workload has not reached the socket memory bandwidth limits. SPECint `hmmer` is an example of this workload class for all simulated thread counts.
2. **Flat**: The workload did not increase or decrease throughput with the addition of threads. This represents a workload that is limited by the socket memory bandwidth, but the reduced effective cache size for each thread did not increase the miss rate. SPECint `libquantum` is an example of this case for all thread counts.
3. **Decreasing**: The socket throughput decreased with the addition of execution threads. These workloads are memory bandwidth bound, and the reduced effective cache size caused a data structure to overrun such that an increase in miss rate is observed. SPECint `mcf` and Fspec `zeusmp` are *decreasing* workloads at 16 threads and 32 threads, respectively.

Note that, once workloads leave region 1, they will alternate between regions 2 and 3. This property is useful in that the maximum throughput is detectable in the transition out of region 1.

Table 3 and Table 4 show the calculated scaling characteristics of the integer and floating point workloads. The regions are highlighted in the tables as shown below,:

region 1	region 2	region 3

The cache footprint of many integer workloads in the SPEC CPU 2006 suite is small enough such that the workloads scale up to the 128 threads. These workloads (`gobmk`, `h264ref`, `hmmer`, `perlbench`, and `sjeng`) are effectively CPU bound. In contrast, the `mcf` and `libquantum` workloads reach a peak throughput at only 16 threads. Most notably, `mcf` has a significant decrease in throughput.

The floating point benchmarks exhibit much larger working sets compared to the integer suite. Only the `gamess` workload scales up to 128 threads. These simulations indicate that the floating point workloads have larger and more varied dataset compared to the integer suite.

	Number Threads							
workload	1	2	4	8	16	32	64	128
astar	1	2	4	8	16	32	49.7	42.6
bzip2	1	2	4	8	16	32	64	70.3
gcc	1	2	4	8	16	32	34.1	32.1
gobmk	1	2	4	8	16	32	64	128
h264ref	1	2	4	8	16	32	64	128
hmmer	1	2	4	8	16	32	64	128
libquantum	1	2	4	8	11.0	11.0	11.0	11.0
mcf	1	2	4	8	16	12.8	9.0	7.7
omnetpp	1	2	4	8	16	32	57.3	52.1
perlbench	1	2	4	8	16	32	64	128
sjeng	1	2	4	8	16	32	64	128
xalancbmk	1	2	4	8	16	32	64	98.9
GeoMean	1	2	4	8	15.5	27.1	43.2	58.0

Table 3: SPEC integer 2006 scaling results

	Number Threads							
workload	1	2	4	8	16	32	64	128
bwaves	1	2	4	8	9.7	9.6	9.3	9.1
cactusADM	1	2	4	8	16	32	28.6	13.7
calculix	1	2	4	8	16	32	44.5	39.4
dealII	1	2	4	8	16	32	32.3	29.1
gamess	1	2	4	8	16	32	64	128
gemsFDTD	1	2	4	8	16	22.1	21.4	20.2

Table 4: SPEC Floating point 2006 scaling results

	Number Threads							
workload	1	2	4	8	16	32	64	128
gromacs	1	2	4	8	16	32	64	67.5
lbm	1	2	4	8	8.6	5.4	5.4	5.2
leslie3d	1	2	4	8	16	26.5	25.7	21.2
milc	1	2	4	8	16	17.5	17.4	17.4
namd	1	2	4	8	16	32	64	49.0
povray	1	2	4	8	16	32	64	38.2
soplex	1	2	4	8	12.9	12.0	11.5	10.9
sphinx3	1	2	4	8	16	30.1	28.4	26.7
tonto	1	2	4	8	16	32	64	72.7
wrf	1	2	4	8	16	32	50.4	44.8
zeusmp	1	2	4	8	16	32	44.0	26.2
GeoMean	1	2	4	8	14.8	23.6	30.3	27.1

Table 4: SPEC Floating point 2006 scaling results

4.0 System Measurements

The analysis in the previous section is based on the specific assumptions that limit system throughput. Specifically, rate throughput is limited by memory bandwidth as the number of contexts grows. In this section we compare hardware benchmark results against the projected results. If the characteristics on the system measurements match those of the simulation results across the mix of workloads, then we show that the assumptions of the model are relevant with respect to actual system behavior.

For the system results to be useful for this correlation, they must represent multiple threads that exhibit similar constraints. Specifically, multiple threads must share a last level cache and the same memory interface. For these systems, we obtained SPEC CPU 2006 results for a range of active thread counts.

The most obvious candidate for correlation is the Sun UltraSparc T1 system [10]. This system has 32 threads sharing a 3MB L2 cache and memory interface. The primary drawback of this system in this context is the low instruction throughput of individual cores. The performance of each thread may fall short of the model assumptions. This effect is containable, in that the inflection point could simply move to a higher thread count. The inherent mechanism of the model would still dominant once the socket bandwidth limit has been reached. In addition, the T1 has a single shared floating point unit for all 8 cores. Therefore we only run the integer workloads. The recently announced T2 [11] version does have floating point units dedicated to each core, but that hardware is not currently available in our lab.

For our T1 benchmark runs we used the following system configuration shown in Table 5.

Parameter	**Value**
Operating System	SunOS 5.10 Generic_118833-17
CPU Frequency	1 GHz
Main Memory	8 GB DDR2 DRAM
Compiler	Sun Studio 11

TABLE 5. UltraSparc T1 System Configuration

Measured T1 scaling results are shown below in Table 6. We were unable to run all 32 thread tests due to system memory constraints on our test system. but an important point with regard to high thread count systems is that high memory capacity is needed. The new Sun T2 supports this larger memory capacity.

	Active Thread Count							
Workload	**1**	**2**	**4**	**6**	**8**	**12**	**16**	**32**
perlbench	1.00	1.99	3.98	5.96	7.47	8.80	7.41	
bzip2	1.00	1.97	3.79	5.55	5.39			
gcc	1.00	1.96	3.77	5.47	5.66			
mcf	1.00	1.82	3.39	4.86	3.32			
gobmk	1.00	2.00	3.99	5.96	7.92	10.78	13.70	20.17
hmmer	1.00	2.00	3.98	5.97	7.94			
sjeng	1.00	2.00	3.99	5.97	7.96			
libquantum	1.00	2.01	3.97	5.91	7.81	11.20	14.50	23.36
h264ref	1.00	2.00	3.97	5.93	7.85			
omnetpp	1.00	1.91	3.70	5.47	7.19	9.79	12.27	14.33

TABLE 6. Measured T1 SPECINT scaling

	Active Thread Count							
Workload	**1**	**2**	**4**	**6**	**8**	**12**	**16**	**32**
astar	1.00	1.96	3.80	5.58	7.30	9.77	9.45	
xalancbmk	1.00	1.91	3.72	5.53	7.31			
GM	1.00	1.96	3.83	5.67	6.75			

TABLE 6. Measured T1 SPECINT scaling

In addition to the T1 system, we also compare submitted results from the multi-core Intel Xeon Core2 based systems. The Xeon systems have a maximum of 4 threads per socket. We do not expect to see decreased throughput with only 4 threads, but the data is useful in validating the general behavior across an additional architecture. We chose the results published by Fujitsu Siemens Computers [22] because the compiler and system configuration was kept effectively constant across the 1, 2 and 4 core results for integer and floating point runs.

The Xeon based Fujitsu Siemens Computers system configuration is shown below in Table 7.

Parameter	**Value**
Operating System	64-Bit SUSE LINUX Enterprise Server 10, Kernel 2.6.16.21-0.8-smp on an x86_64
CPU Frequency	2.667 GHz
Main Memory	16 GB (8x2 GB DDR2 PC2-5300F, 2 rank, CAS 5-5-5)
Compiler	Intel C++ Compiler for IA32/EM64T application, Version 9.1 - Build 20070215, Package-ID: l_cc_p_9.1.047 Intel Fortran Compiler for IA32/EM64T application, Version 9.1 - Build 20070215, Package ID: l_fc_p_9.1.043

TABLE 7. Fujitsu Siemens Xeon system configuration.

Figure 1 shows the Xeon published SPECint scaling results. The five benchmarks (gobmk, h264ref, hmmer, perlbench, and sjeng) shown to scale well in Table 3 mimic the same linear characteristic as the Xeon results. This is due to the low cache miss rates.

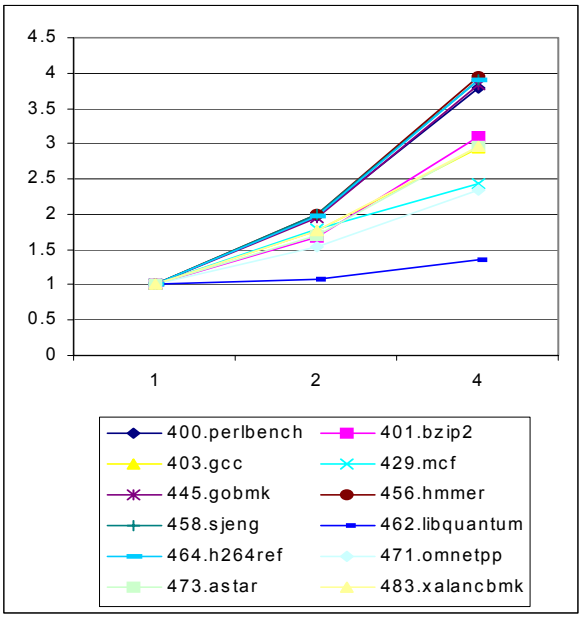

FIGURE 1. Xeon SPECint scaling on 2.667 GHz Fujitsu Siemens system

Figure 2 shows the Xeon published SPECfp scaling results. A large fraction of the results exhibit scaling limitations, which matches the simulation data.

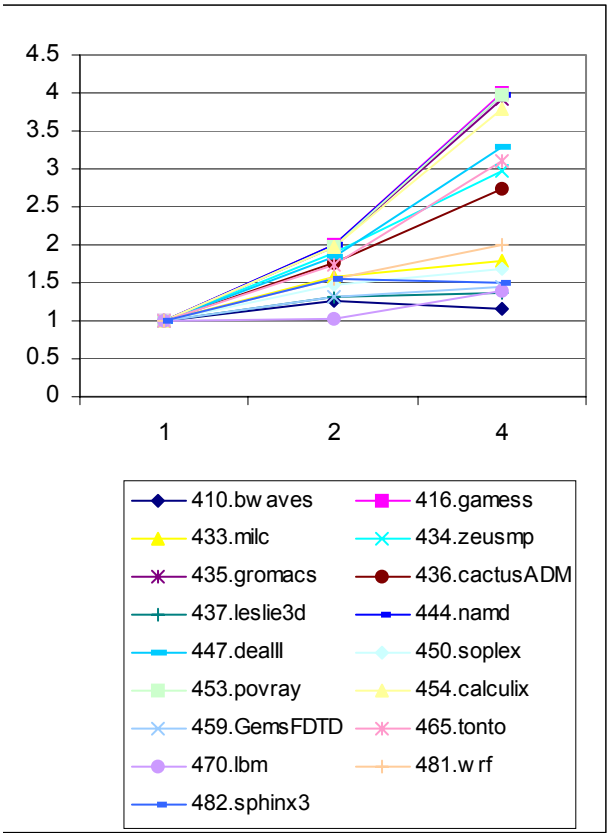

FIGURE 2. Xeon SpecFp scaling on 2.667 GHz Fujitsu Siemens system

5.0 Detailed Benchmark Analysis

A key aspect of this research is identifying the variation in the benchmarks characteristics that would warrant more sophisticated system optimization in thread scheduling algorithms. The following analysis of an interesting set of benchmark codes was based on instruction profiling using the tprof tool. The tprof tool guided source code investigation by revealing the *hot* regions of the benchmarks.

5.0.1 Ispec.sjeng (easy cache)

`Sjeng` is an example workload that has a very small cache footprint. The miss rate remains constant as the cache per thread decreases from 4 MB down to 32kB. Therefore, additional execution resources do not stress the available socket memory bandwidth. Both the Xeon and T1 systems exhibited very close to ideal scaling as shown in Table 8.

	Number Threads				
	1	2	4	6	8
Expected Scaling	1	2	4		8
T1	1	2	3.99	5.97	7.96
Xeon	1	2.01	3.90		

TABLE 8. Sjeng scaling

5.0.2 Ispec.libquantum (steady cache)

Analysis of `libquantum` tprof data showed that most of the execution time is spent sequentially operating on one large 32 MB array. This 32 MB array contains a structure with two 8 byte elements. Only one of these elements is frequently updated, therefore only half of the array data is actually referenced.

This following code segment shows the structure of each array element.

```
struct quantum_reg_node_struct
{
  COMPLEX_FLOAT amplitude; /* alpha_j */
  MAX_UNSIGNED state;      /* j */
};
```

The code segment below contains the loop where ~70% of execution time is spent. An additional ~15% is spent in code with the same data access pattern. The dominant memory access is to the 'reg->node[i].state' element. All misses to this structure are due to *Capacity* misses.

```
for(i=0; i<reg->size; i++)
  if(reg->node[i].state & (1<<control1))
    if(reg->node[i].state & (1<<control2))
      reg->node[i].state ^= (1<<target);
```

Since the largest cache capacity per thread simulated is only 4MB, libquantum garners only spatial locality from the caches. The lower cache capacity per thread has no affect on the cache miss rates. This explains the flat scaling profile of the benchmark in the simulated results. Adding threads shows a gain up until memory bandwidth saturation.

The two hardware measurements show substantially different results (Figure 3). The Xeon system shows meager gains. Conversely, the Sun T1 result shows increasing throughput linearly up to 20 threads. The source of this discrepancy lies in the miss access pattern. The sequential access sequence to '`node[i].state`' is highly prefetchable by the Xeon hardware facility. The T1 does not contain prefetch hardware because it is designed to tolerate memory misses through large numbers of threads. Interestingly, the peak T1 result increases up to 32 threads,

exceeding the Xeon result by 67%. The T1 machine has a claimed memory bandwidth of 20 GB/sec compared with only 5.8 GB/sec measured on a Xeon system. Therefore, the higher result on this memory bandwidth dominated benchmark is expected.

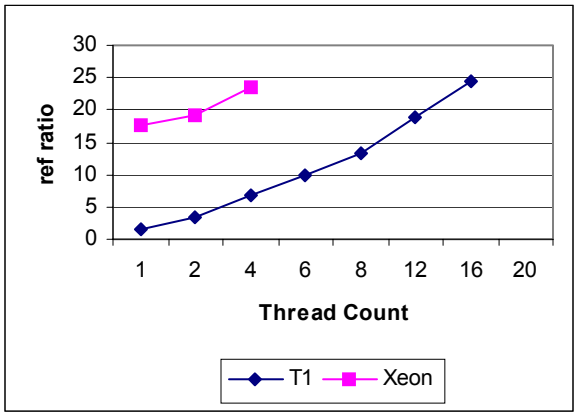

FIGURE 3. libquantum throughput comparison between Sun T1 and Intel Xeon systems.

5.0.3 Ispec.mcf (negative returns)

The mcf benchmark operates on a complex and large dataset relative to the other integer benchmarks. The algorithm solves an optimization problem with successive steps alternating between two routines [24]. In addition, the amount of data accessed in each iteration is data dependant and varies between iterations. This non-cache resident access pattern produces a steady decrease in hit rate as the effective cache size for the workload is reduced. This increasing miss rate characteristic causes a decrease in throughput as threads are added once the memory bandwidth threshold is reached. The throughput for the three system types is shown in Figure 4. The simulated result is based on 20 GB/sec of memory bandwidth in order to match the T1 capabilities. The simulated mcf throughput peaks at 9.3 instructions/clock for a 16 thread system, dropping to 6.4 instructions/clock for a 32 thread systems. This behavior is also seen in the Sun T1 system runs. The T1 throughput peaks at 6 threads of execution with a throughput of 4.85 times the single threaded result. The addition of two more threads decreases throughput to 3.22. Additional thread context beyond 8 resulted in page misses due to the limited amount of system memory in our T1 test system.

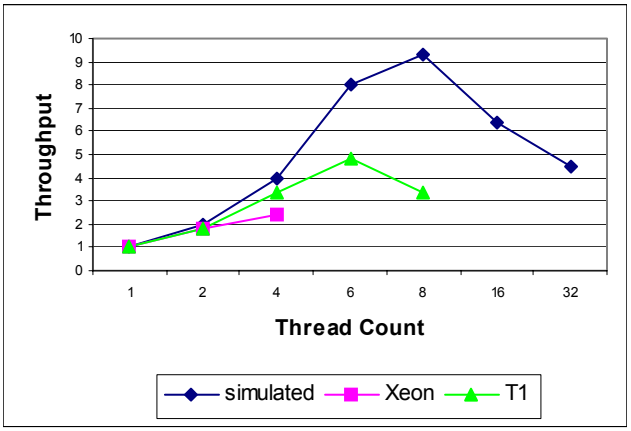

FIGURE 4. mcf scaling results across simulation projections and the measures T1 and Xeon results

The close thread peak correlation between the simulated and T1 systems provides good insight into the validity of the simulation assumptions for this workload. The differences in magnitude of the results are due mainly to the differences in the baseline throughput of the two systems.

6.0 System Optimization

The key observation of this analysis is rooted in diminishing and negative returns with the addition of execution threads. More importantly, the point at which this occurs is variable between workloads. If this point was common across workloads, a designer would simply build that ideally sized system. If the system is designed around the worst case workload (lowest thread scalability), we cannot take advantage of the more cache friendly workloads' scalability. Current computer systems do not take this effect into account. Today's systems utilize a greedy approach, where as many threads as possible are dispatched. This provides an opportunity for optimization. If the hardware limits the number of available threads for the less scalable workloads, performance is not optimal for more scalable workloads.

The most basic optimization would be to monitor IPC across the threads. The scheduler would dispatch additional threads up until the IPC failed to increase. This technique is somewhat useful, but fails to monitor the memory bandwidth utilization, which is the root cause driving the throughput limits. This becomes more important in the optimization of a mixed set of workloads. Simply trying different combinations of threads and execution counts would be invasive and unlikely to converge.

We propose a more sophisticated system where the memory bandwidth utilization of each thread is monitored. With this information intelligent scheduling, decisions can be made such that core execution and available memory bandwidth can be effectively utilized.

6.1 Bandwidth Sensitivity

The primary simulation analysis was based on an assumption of 40 GB/sec of socket memory bandwidth. In addition, a sensitivity study across a range of bandwidths was evaluated. Results are shown in Table 9 and Table 10. Bandwidths from 5 GB/sec to 50 GB/sec in 5 GB/sec increments were used. A speedup was calculated for the range of socket bandwidths for the SPEC CPU2006 integer and floating point suites. The speedup compares a system with dynamic thread dispatch control to a greedy dispatch where all available threads are used. For a system with lower memory bandwidth the speedup reaches 1.54 and 2.17 on the integer and floating point results.

	maximum number threads in system							
bandwidth	1	2	4	8	16	32	64	128
5 GB/sec.	1	1	1	1.01	1.03	1.12	1.23	1.54
10 GB/sec.	1	1	1	1	1.02	1.06	1.15	1.33
15 GB/sec.	1	1	1	1	1.01	1.05	1.12	1.24
20 GB/sec.	1	1	1	1	1	1.03	1.10	1.20
25 GB/sec.	1	1	1	1	1	1.03	1.07	1.14
30 GB/sec.	1	1	1	1	1	1.03	1.07	1.14
35 GB/sec.	1	1	1	1	1	1.03	1.07	1.12
40 GB/sec.	1	1	1	1	1	1.02	1.05	1.10
45 GB/sec.	1	1	1	1	1	1.01	1.04	1.09
50 GB/sec.	1	1	1	1	1	1	1.03	1.07

Table 9: Integer Dynamic threads speedup vs socket bandwidth

	maximum number threads in system							
bandwidth	1	2	4	8	16	32	64	128
5 GB/sec.	1	1	1.01	1.02	1.09	1.26	1.40	2.17
10 GB/sec.	1	1	1	1	1.05	1.20	1.31	1.91
15 GB/sec.	1	1	1	1	1.01	1.10	1.20	1.64
20 GB/sec.	1	1	1	1	1	1.08	1.15	1.53
25 GB/sec.	1	1	1	1	1	1.07	1.14	1.47
30 GB/sec.	1	1	1	1	1	1.05	1.11	1.40
35 GB/sec.	1	1	1	1	1	1.04	1.09	1.35
40 GB/sec.	1	1	1	1	1	1.03	1.07	1.31
45 GB/sec.	1	1	1	1	1	1.03	1.07	1.28
50 GB/sec.	1	1	1	1	1	1.03	1.07	1.24

Table 10: Floating point Dynamic threads speedup vs socket bandwidth

6.2 Power Implications

The dynamic thread dispatch policy provides for ideal throughput in a highly threaded design through tuning the number of thread contexts to most efficiently utilize system resources. The negative implications of ideal thread contexts must be considered in the system's design. Hardware design can provide power control capabilities for both active and static power consumption. Static power of the leaky unused thread context and cores can be reduced by independent voltage control[23] or power gating methods [9]. Active power can be reduced by decreasing clock rates on memory bandwidth dominated workloads, such that just enough traffic is generated to sustain bandwidth.

7.0 Conclusions and Future Work

We use simulation and analysis to investigate the inflection point at which additional thread contexts begin to decrease throughput and power efficiency. The inflection point was found to vary significantly depending on the workload. The results indicate performance can be improved by monitoring IPC across threads and scheduling memory bandwidth consumption across threads. The results also indicate that hardware control mechanisms and decreased clock rates can improve power/performance. Future work includes detailed simulation and power models to quantify the observed effects for a spectrum of processor tradeoffs.

8.0 References

[1] P. Vogt, "Fully buffered dimm (fb-dimm) server memory architecture: Capacity, performance, reliability, and longevity.". Intel Developer Forum, February 2004.

[2] Brinda Ganesh, Aamer Jaleel, David Wang, and Bruce Jacob. "Fully-Buffered DIMM memory architectures: Understanding mechanisms, overheads and scaling.". Proc. 13th International

Symposium on High Performance Computer Architecture (HPCA 2007). Phoenix AZ, February 2007.

[3] Peter Benkart, Alexander Kaiser, Andreas Munding, Markus Bschorr, Hans-Joerg Pfleiderer, Erhard Kohn, Arne Heittmann, Holger Huebner, Ulrich Ramacher, "3D Chip Stack Technology Using Through-Chip Interconnects," IEEE Design and Test of Computers, vol. 22, no. 6, pp. 512-518, Nov/Dec, 2005

[4] Li, F., Nicopoulos, C., Richardson, T., Xie, Y., Narayanan, V., and Kandemir, M. 2006. Design and Management of 3D Chip Multiprocessors Using Network-in-Memory. SIGARCH Comput. Archit. News 34, 2 (May. 2006), 130-141.

[5] Li, Y.L.; Figueroa, D.G.; Chickamenahalli, S.A.; Chee Yee Chung; Teong Guan Yew; Cornelius, M.D.; Do, H.T. Enhancing power distribution system through 3D integrated models,optimized designs, and switching VRM model. Electronic Components and Technology Conference, 2000. 2000 Proceedings. 50th. Volume , Issue , 2000 Page(s):272 - 277

[6] Beckmann, B. M. and Wood, D. A. 2004. Managing Wire Delay in Large Chip-Multiprocessor Caches. In Proceedings of the 37th Annual IEEE/ACM international Symposium on Microarchitecture (Portland, Oregon, December 04 - 08, 2004). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, 319-330.

[7] Black, B., Nelson, D. W., Webb, C., and Samra, N. 2004. 3D Processing Technology and Its Impact on iA32 Microprocessors. In Proceedings of the IEEE international Conference on Computer Design (Iccd'04) - Volume 00 (October 11 - 13, 2004). ICCD. IEEE Computer Society, Washington, DC, 316-318.

[8] D. M. Berger, J. Y. Chen, F. D. Ferraiolo, J. A. Magee, G. A. Van Hubert. High-speed source-synchronous interface for the IBM System z9 processor. January 2007 IBM Journal of Research and Development, Volume 51 Issue 1

[9] Suhwan Kim , Stephen V. Kosonocky , Daniel R. Knebel , Kevin Stawiasz. Experimental measurement of a novel power gating structure with intermediate power saving mode, Proceedings of the 2004 international symposium on Low power electronics and design, August 09-11, 2004, Newport Beach, California, USA

[10] Sun Microsystems Inc. UltraSPARC T1 Overview. http://www.sun.com/processors/UltraSPARC-T1/, Dec. 2005.

[11] Sun Mircosystems Inc. UltraSPARC® T2 Processor. http://www.sun.com/processors/UltraSPARC-T2/, Aug 2007.

[12] Intel Corporation. Quad-Core Intel Xeon Processor 5300 Series. http://download.intel.com/products/processor/xeon/dc53kprodbrief.pdf

[13] Intel Corporation. Dual-Core Intel® Itanium® 2 Processor 9000 Series. http://download.intel.com/products/processor/itanium2/dc_prod_brief.pdf

[14] Rusu, S. Muljono, H. Cherkauer, B. Itanium 2 processor 6M: higher frequency and larger L3 cache Intel Corp., Santa Clara, CA, USA; IEEE Micro Mar-Apr 2004 Volume: 24, Issue: 2 page(s): 10- 18

[15] AMD. Introducing Barcelona. http://multicore.amd.com/us-en/AMD-Multi-Core/Products/Barcelona.aspx

[16] Power6. IBM. http://www-03.ibm.com/press/us/en/presskit/21546.wss

[17] Gerhard Schrom, Peter Hazucha, Jae-Hong Hahn, Volkan Kursun, Donald Gardner, Siva Narendra, Tanay Karnik, Vivek De. Low power converter circuits: Feasibility of monolithic and 3D-stacked DC-DC converters for microprocessors in 90nm technology generation. Low Power Electronics and Design, 2004. ISLPED apos;04. Proceedings of the 2004 International Symposium on. Volume , Issue , 9-11 Aug. 2004 Page(s): 263 - 268

[18] B. Sinharoy , R. N. Kalla , J. M. Tendler , R. J. Eickemeyer , J. B. Joyner, POWER5 System microarchitecture, IBM Journal of Research and Development, v.49 n.4/5, p.505-521, July 2005

[19] Sally A. McKee, Reflections on the memory wall, Proceedings of the 1st conference on Computing frontiers, April 14-16, 2004, Ischia, Italy

[20] Standard Performance Evaluation Corporation. http://www.specbench.org.

[21] J. McCalpin. Stream: Sustainable memory bandwidth in high performance computers. http://www.cs.virginia.edu/stream/, 1999.

[22] SpecCpu2006. http://www.spec.org/cpu2006/

[23] Tse-Yu Yeh. The Low-Power High-Performance Architecture of the PWRficient Processor Family, P.A. Semi Proceedings of the 18th HotChips Symposium, Aug. 2006.

[24] Andreas Lobel. MCF version 1.2. http://www.zib.de/Optimization/Software/Mcf/latest/mcf-1.2.pdf

[25] Sherwood, T., Perelman, E., Hamerly, G. and Calder, B. Automatically Characterizing Large Scale Program Behavior. International Conference on Architectural Support for Programming Languages and Operating Systems, October 2002.

[26] Yue Luo, Lizy K. John, and Lieven Eeckhout. Self-Monitored Adaptive Cache Warm-Up for Microprocessor Simulation. 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). October 2004

[27] H. Jacobson, P. Bose, Z. Hu, A. Buyuktosunoglu, V. Zyuban, R. Eickemeyer, L. Eisen, J. Griswell, D. Logan, B. Sinharoy and J. Tendler, “Stretching the Limits of Clock-Gating Efficiency in Server-Class Processors,” Proceedings of the International Symposium on High-Performance Computer Architecture, February 2005.

[28] I. Hur and C. Lin, “Adaptive History-Based Memory Schedulers,” Proceedings of the International Symposium on Microarchitecture, 2004.

[29] J. M. Borkenhagen, R. J. Eickemeyer, R. N. Kalla and S. R. Kunkel, “A Multithreaded PowerPC Processor for Commercial Servers,” IBM J. of Res. Dev., Vol. 44 No. 6, November 2000

[30] S. R. Kunkel, R. J. Eickemeyer, M. H. Lipasti, T. J. Mullins, B. O’Krafka, H. Rosenberg, S. P. VanderWiel, P. L. Vitale and L. D. Whitley, “A Performance Methodology for Commercial Servers,” IBM J. Res. & Dev. Vol. 44, No. 6, November 2000, pp. 851-872.