

# Scaling Up the Atlas Chip-Multiprocessor

Peter G. Sassone, *Student Member, IEEE*, and  
D. Scott Wills, *Senior Member, IEEE*

**Abstract**—Atlas, a dynamically multithreading chip-multiprocessor (CMP), gains little complexity as processing elements are added. When the model is scaled up with strategic layouts and realistic latencies, area and power efficiency surpass that of an aggressive out-of-order processor, though results are sensitive to global communication delay.

**Index Terms**—Dynamic multithreading, chip-multiprocessor, scaling.

## 1 INTRODUCTION

AS modern out-of-order (OOO) processors widen, deepen, and become more aggressively issued, their complexity grows faster than the marginal performance improvements. One primary issue is extracting parallelism in a timely manner—a difficult task with a single application. After exhausting the data and instruction level parallelism in these binaries, architects have recently turned to dynamically extracting thread level parallelism (TLP) for speedup. These proposals explore the creation of threads in hardware to extract TLP without compiler or programmer support [1], [2], [3], [4], [5]. Most of these techniques, however, only increase the complexity of conventional out-of-order processors further. Only some of this prior work utilizes chip-multiprocessors for dynamic multithreading despite their simpler design, easier validation, and shorter wire lengths [6].

In previous publications, Atlas was presented as a clever and effective way of using an eight-way chip-multiprocessor (CMP) with dynamic multithreading to speed up sequential applications [7], [8]. As transistor integration grows toward one billion transistors on a chip, however, the question of Atlas scalability emerges: Though a chip with more processing elements is technically feasible, do silicon and power efficiency make such designs desirable?

To explore this issue, we create simple guidelines for Atlas configurations with various numbers of processing elements. These configurations are then analyzed with various architectural tools to extract design parameters, which we then use to derive global latencies and frequencies. Feeding these latencies into a cycle accurate simulator produces execution throughput results at various scaling points. We then analyze these configurations on area and power efficiency metrics to judge the practicality of such designs. Finally, we vary key assumptions to determine the sensitivity of results to these parameters.

## 2 RELATED WORK

Dynamic multithreading on chip-multiprocessors is a relatively new subject in computer architecture. Despite the design simplicity of CMPs, the challenge of adequately supplying each processing element with useful work from a single-threaded binary is formidable. Researchers have proposed many widely varying

solutions; this is understandable considering that issues such as task partitioning are NP-complete [9].

One such solution is Dynamic Multithreading Processors [1], which use loop boundaries to divide the application stream. Multiscalar [4] divides the instructions based on basic block boundaries and alleviates data dependencies with an internode ring for passing data values, similar to Atlas. Since loops and blocks can be of any length, Trace Processors [3] introduced the idea of fixed-length threads to balance the workload between processors and keep thread-size to a reasonable number. Slip-stream Processors [10] don't divide the instruction stream up at all, but rather run two copies of the same program on two processing elements and use results from one to help the other along. As a variation on the idea, Chappell et al. first introduced the idea of creating helper threads dynamically to assist a primary thread [5].

By buffering threads' state until they become nonspeculative, most of these proposals enable deep speculation without complex data versioning. Many solutions, including Trace, successfully use data prediction to greatly reduce the overhead of interthread dependencies. The simplicity of these proposals' value prediction, however, is an impediment to further speedup which Atlas addresses.

Researchers at Carnegie Mellon University have also looked at how speculative multithreading scales with CMPs [11]. They show that adding nodes produces a diminishing effect on speedup and, at six to eight processors, execution times actually begin to rise. As the probability of interthread dependency violations is proportional to the number of computing nodes in a system, scaling up the CMP produces more violations and, thus, slower execution rates. However, since this experiment was targeted at conventional crossbar CMP designs without value prediction, their results do not necessarily predict Atlas scalability.

## 3 ATLAS BACKGROUND

Prior work has presented Atlas as an effective use of a modified chip-multiprocessor (see Fig. 1) to speed up sequential applications [7]. The design is most similar to the Multiscalar CMP [4], but with two critical distinctions. First is Atlas's novel mem-slicing algorithm, which divides the instruction stream at memory references [8] instead of branches. Second is the use of an aggressive combined value and control predictor described below. Thread partitioning is completely dynamic and works only by looking at the instructions in the execution window. As a new thread is created, it is issued to the next node (processing element, or PE) in the ring if it is empty. If not, the processor stalls until that node is free. Only one node at any time is nonspeculative, indicated with a rotating token. Speculative nodes buffer their state until they are marked as nonspeculative or their threads are quashed [4].

Each node is a simple, five-stage, in-order processor with 64 KB of L1 cache. Out-of-order execution is achieved by these nodes working in parallel, so an elementary design is sufficient and easy to verify. Two shared buses are attached to each node—one to the L2 cache and the other to the value/control predictor. Similar to Multiscalar, the nodes are also interconnected with a pipelined, bidirectional ring which provides data forwarding. A ring provides scalability while still being practical as threads communicate most frequently with their nearest-neighbor threads.

Global control is maintained in the shared value/control predictor. This same predictor is also used to track and predict interthread register and memory dependencies. This innovative unified design uses the same tables for predicting data values and branch outcomes as the latter is merely a subset of the former. This structure is highly effective for Atlas, allowing for deep speculation to extract thread-level parallelism. In the case of errant

• The authors are with the Microelectronics Research Center, Electrical and Computer Engineering, Georgia Institute of Technology, 777 Atlantic Dr., Atlanta, GA 30332-0250. E-mail: {scott.wills, sassone}@ece.gatech.edu.

Manuscript received 3 Apr. 2003; revised 23 Apr. 2004; accepted 26 July 2004; published online 16 Nov. 2004.

For information on obtaining reprints of this article, please send e-mail to: [tc@computer.org](mailto:tc@computer.org), and reference IEEECS Log Number 118531.

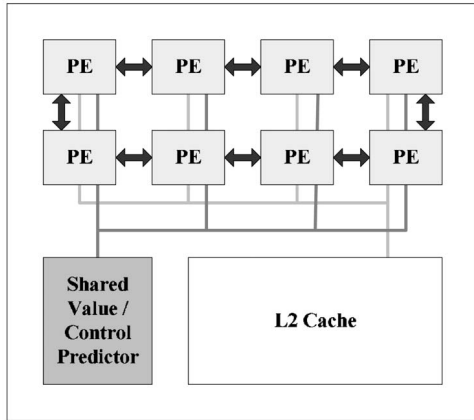


Fig. 1. Atlas CMP architecture block diagram.

prediction, however, each processor's instruction queue has been modified to provide fine-grained recovery—only the instructions dependent on the value misspeculation will be quashed and reexecuted. Each control misprediction, though rare, requires quashing all dependent threads and, thus, is quite expensive.

In a typical out-of-order processor, instruction scheduling and issue is quite complex and thus becomes an execution bottleneck. Atlas's centralized thread dispatch is very efficient, on the other hand, leaving the fine-grain issue to the nodes. As a result of this two-phase issue, centralized thread delimitation is not considered a significant source of delay even with high instruction throughput.

#### 4 PHYSICAL DESIGN

Previous publications focused on an eight-node configuration of Atlas (as shown in Fig. 1) with an L2 cache hit-latency of four-cycles and value/control predictor latency of one cycle. However, neither the design nor the simulator are restricted to these parameters. We wish to analyze how Atlas performs with various numbers of processing elements and realistic global delays. Since we have eliminated the central crossbar present in many chip-multiprocessors, the only global delays are the access times to L2 and the value/control predictor.

First, to determine these latencies, accurate physical floorplans must be developed. To simplify the layout process, we have developed simple heuristics for Atlas configurations:

- Use a grid layout for all elements.
- Minimize the area of the design.
- Create a square final layout.
- Minimize the distance to the global structures (L2 and value/control predictor).
- Maintain tight adjacencies in the ring of nodes.

The grid cell size is the size of a single node. GENESYS [2], a tool developed at Georgia Tech for integrated circuit area and clocking analysis, estimates 1.1 million transistors per node occupying an area of  $4.38\text{mm}^2$  and using  $3.67\text{W}$  (40 percent of which is static power) on a 65nm process. This creates grid cells of roughly  $2.1\text{mm}$  on each edge. This grid will also accommodate the value/control predictor and the L2 cache, so the sizes of each must also be determined in terms of grid cells. For the predictor, we use the safe assumption that it is no bigger than a node and, thus, will occupy one cell. The L2 is best left as the excess contiguous grid cells on the die, assuming it is a reasonable size.

Fig. 2 shows Atlas sample layouts for node counts from 4 to 64. Though these are not the only way to arrange these elements, they have been found to be optimal given the guidelines above. The

dotted lines represent the data-forwarding ring between adjacent nodes. All layouts have the value/control predictor (VP) and between 1.25 and 2.75 megabytes of L2 cache. These cache sizes were determined via CACTI [12] with 64-byte line sizes and 8-way associativity for caches less than 2MB and 16-way for caches larger than 2MB. Though the Atlas simulator models a perfect L2, experiments (not shown for brevity) demonstrate that these L2 sizes are sufficient to eliminate nearly all noncompulsory misses in the benchmarks.

Each layout in Fig. 2 also shows the global statistics, such as the number of nodes and total area in square millimeters. Also displayed is the total peak power estimate (dynamic plus static), which is the sum of the node power estimates and the cache estimate. The former is the power usage of a busy node, though it is likely in larger configurations that not all nodes are busy. The cache power estimate is from CACTI, assuming one access per node cycle (2.5 billion transactions per second). This power estimate also assumes that each node receives a clock signal on a dedicated pin and, thus, no global clock tree is needed. Finally, Fig. 2 shows the average number of grid cell hops between nodes and global structures using Manhattan routing (up/down/left/right). The average VP path is the average number of hops to the value/control predictor cell and the average hops to the farthest grid cell of cache is the L2 path.

The largest design in Fig. 2 with 64 nodes uses approximately 74 million logic transistors. This compares well with modern microprocessors, such as the 130nm Intel Itanium 2, which uses 410 million logic transistors [13]. Perspective for the power and area of the 64-node configuration comes from the 2003 International Technology Roadmap for Semiconductors (ITRS) [14]. Their forecasts show that high-volume microprocessors at 65nm are projected to have an area of  $280\text{mm}^2$  and consume  $104\text{W}$  at introduction. Though the  $239\text{W}$  power estimate of the 64-node configuration is for all nodes active at once, it is probably beyond mass manufacturability at 65nm. However, as the results will show, integer applications do not effectively utilize this large arrangement.

#### 5 TIMING EXTRACTION

We now extract timing parameters from each Atlas layout to use in simulation. As mentioned previously, there are two global latencies of importance: value/control predictor and L2 access times. For both of these derivations, total latency is defined as the signal propagation time from the node to the structure, plus the lookup time within that structure, plus the propagation time to return the value to the node.

Utilizing GENESYS, we determine an approximate cycle time for the simple 5-stage nodes of  $0.4\text{ns}$  (2.5GHz frequency) on a 65nm process. This is reasonable as aggressive modern 90nm processors surpass 3 GHz. As this is representative of the time signals take to move across a node, this number is also used as the latency of one grid cell hop. This makes the number of hops computed in Fig. 2 the one-way signal propagation time in node cycles. The value/control predictor is modeled with a lookup time of one cycle, which is added to two times the one-way trip to get a complete round-trip latency. Later results will analyze performance sensitivity to both the predictor lookup delay and one cycles-per-grid-cell delay. The average latencies are shown in Table 1, though simulations will use the actual latency at each node.

CACTI is then used to determine the L2 access times for each cache size. Dividing that number by the cycle time of  $0.4\text{ns}$  yields the node cycles needed for L2 lookups. That time, plus two times the propagation time to L2, produces the total round-trip latency for L2. The average of these values are shown in Table 2. As with the value predictor latency, performance simulations will use the specific L2 latency for each node, not the average.

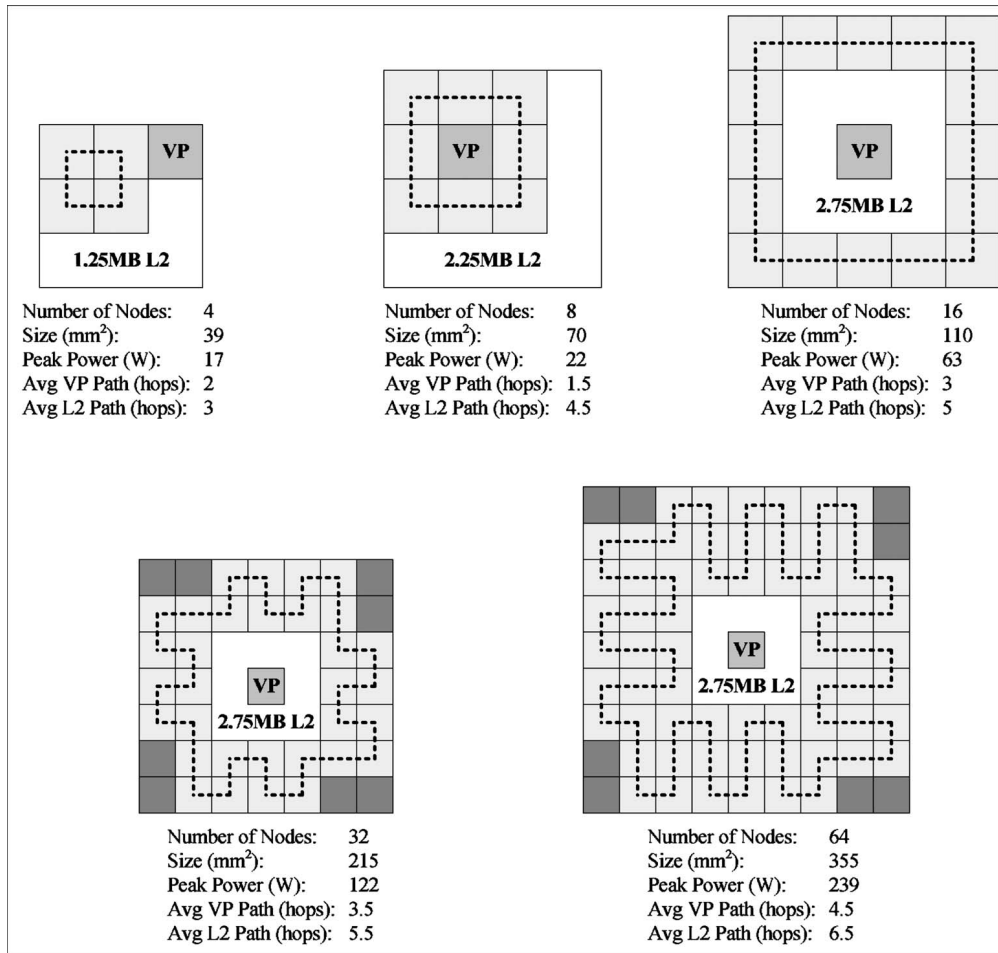


Fig. 2. Atlas configurations with 4, 8, 16, 32, and 64 nodes with statistics for area, power, and average number of cells to global structures.

## 6 EXPERIMENTS AND RESULTS

We now use these latencies and analyze the simulated performance of Atlas as nodes are added. Though these are important factors in performance scaling, the application itself provides the other variable. Prior work shows that potential speedup is limited by five software factors: control mispredictions, value mispredictions, load-balancing, cache misses, and thread creation/retirement overhead. The two most critical factors are control and data misses for Atlas is dependent on the accuracy of such predictions. Cache misses are not as significant in a speculative multithreaded processor since most loads are performed before the thread

becomes nonspeculative. This effectively performs cache-prefetching, hiding most memory penalties [7].

### 6.1 Experimental Setup

To evaluate the performance impact of realistic global latencies, we have modified the SimpleScalar 2.0 [15] cycle-accurate simulator to simulate Atlas in these various configurations. The parameters used are shown in Table 3. The configuration shown is not entirely realistic (i.e., perfect L2 cache), but is sufficient to analyze the scaling characteristics of benchmarks. The relative performance of different node-count designs is independent of L2 hit-rate as long as workloads are not contrived (i.e., 99 percent memory operations)

TABLE 1  
Value/Control Prediction Latencies for Each Configuration

|          | Avg prop hops<br>(from Figure 2) | Avg prop cycles<br>(equal to hops) | Lookup cycles<br>(assumed to be 1) | Total cycles<br>(prop+lookup+prop) |
|----------|----------------------------------|------------------------------------|------------------------------------|------------------------------------|
| 4 nodes  | 2                                | 2                                  | 1                                  | 5                                  |
| 8 nodes  | 1.5                              | 1.5                                | 1                                  | 4                                  |
| 16 nodes | 3                                | 3                                  | 1                                  | 7                                  |
| 32 nodes | 3.5                              | 3.5                                | 1                                  | 8                                  |
| 64 nodes | 4.5                              | 4.5                                | 1                                  | 10                                 |

TABLE 2  
L2Cache Latencies for Each Configuration

|          | Avg prop hops<br>(from Figure 2) | Avg prop cycles<br>(equal to cycles) | Lookup time<br>(from CACTI) | Lookup cycles<br>(time / 0.4 ns) | Total cycles<br>(prop+lookup+prop) |
|----------|----------------------------------|--------------------------------------|-----------------------------|----------------------------------|------------------------------------|
| 4 nodes  | 3                                | 3                                    | 2.11 ns                     | 5                                | 11                                 |
| 8 nodes  | 4.5                              | 4.5                                  | 2.95 ns                     | 7                                | 16                                 |
| 16 nodes | 5                                | 5                                    | 3.47 ns                     | 9                                | 19                                 |
| 32 nodes | 5.5                              | 5.5                                  | 3.47 ns                     | 9                                | 20                                 |
| 64 nodes | 6.5                              | 6.5                                  | 3.47 ns                     | 9                                | 22                                 |

TABLE 3  
Simulated Atlas Configuration

|                           |  |
|---------------------------|--|
| Value/Control Predictor   | 128kb AMA [7]                                |
| Speculative Data Cache    | 32Kb, 2-way associative                      |
| Write Buffer              | 64-entry fully associative                   |
| PE Configuration          | Single issue, in-order, pipelined (5 stages) |
| Issue/Recovery Queue Size | 64 entry                                     |
| Minimum Inst per Thread   | 16   |
| Instruction Cache         | Perfect                                      |
| L2 cache                  | Perfect                                      |

and cache size is sufficient (the configurations tested meet this criteria). Since the L2 is an important piece of any chip-multi-processor, however, we have focused all current research on developing a speculative L2 coherency protocol for Atlas.

Evaluated benchmarks are from SPEC2000int and MediaBench [16]. Any benchmarks not included from these suites did not compile cleanly under gcc 2.95.3 with O2 optimizations. Inputs for SPEC applications come from the test data set and the default Mediabench inputs have been enlarged to lengthen their executions. After skipping the first 100 million instructions, the next 500 million are simulated.

6.2 IPC Results

Fig. 3 shows instructions per cycle (IPC) for the configurations introduced earlier. We also include, for comparison purposes, the execution rate of an aggressive out-of-order uniprocessor similar to the Alpha 21364 (4-wide issue, max 80 instructions in flight, 128 KB L1, 1.5 MB L2). The Mediabench and Spec2000int averages are shown as the last bars in their respective sections.

Scalability ranges from Mediabench’s pegwit-decode, which continues to scale logarithmically even with 64 nodes, to adpcm-decode, which loses performance with every additional node. Multimedia applications, in general, have a higher potential for scaling than conventional integer programs due to the abundant parallelism present. Poor control and value predictability, however, prevent applications such as adpcm-decode from realizing this parallelism across nodes. Though most applications have decent control prediction (94 percent) and data prediction rates (73 percent), most also have limited parallelism and thus find peak performance at an intermediate point such as eight nodes. Spec2000int applications, especially, have limited parallelism and, thus, perform better on traditional out-of-order designs.

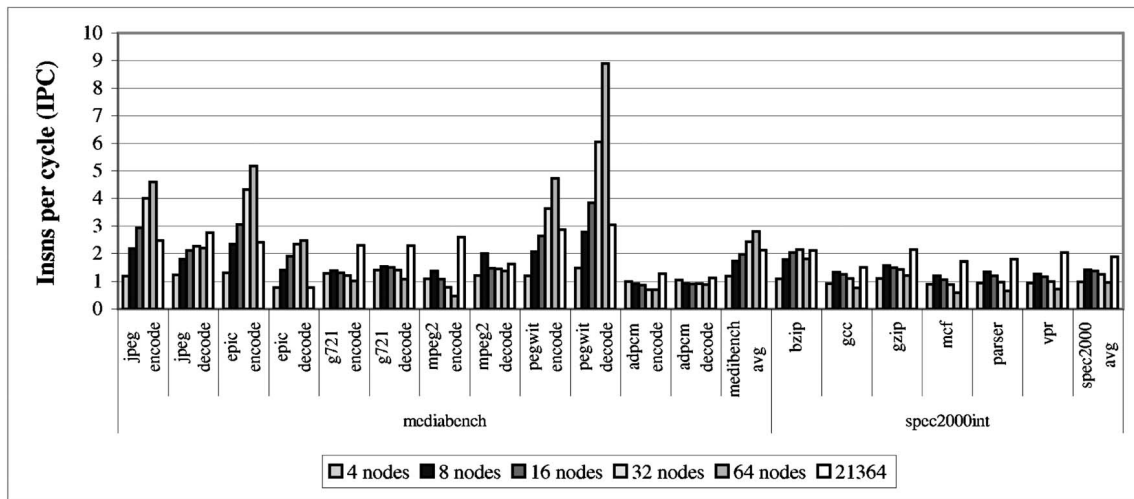


Fig. 3. Instructions per cycle (IPC) for Atlas configurations and out-of-order baseline machine.

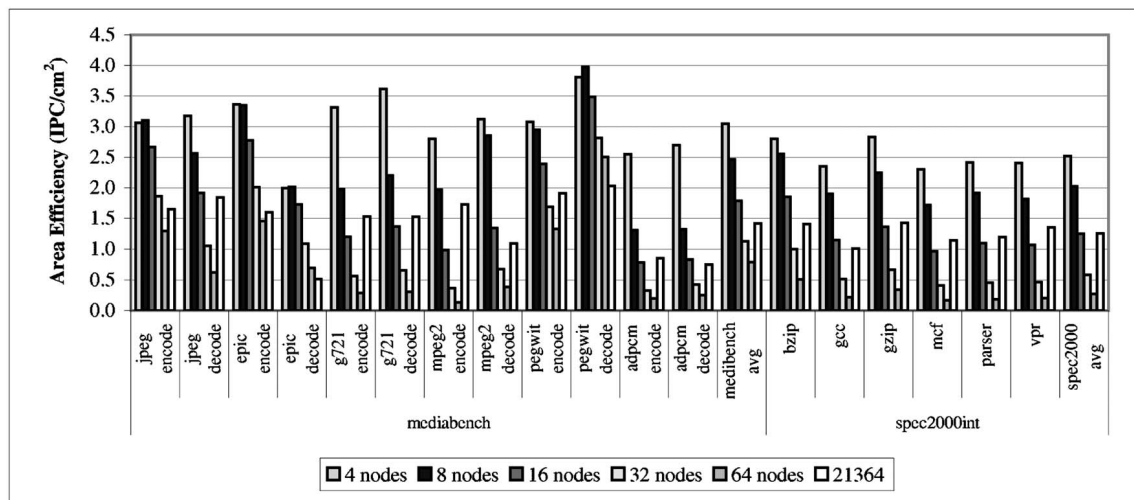


Fig. 4. Silicon efficiency (IPC per cm²) for Atlas configurations and OOO baseline machine.

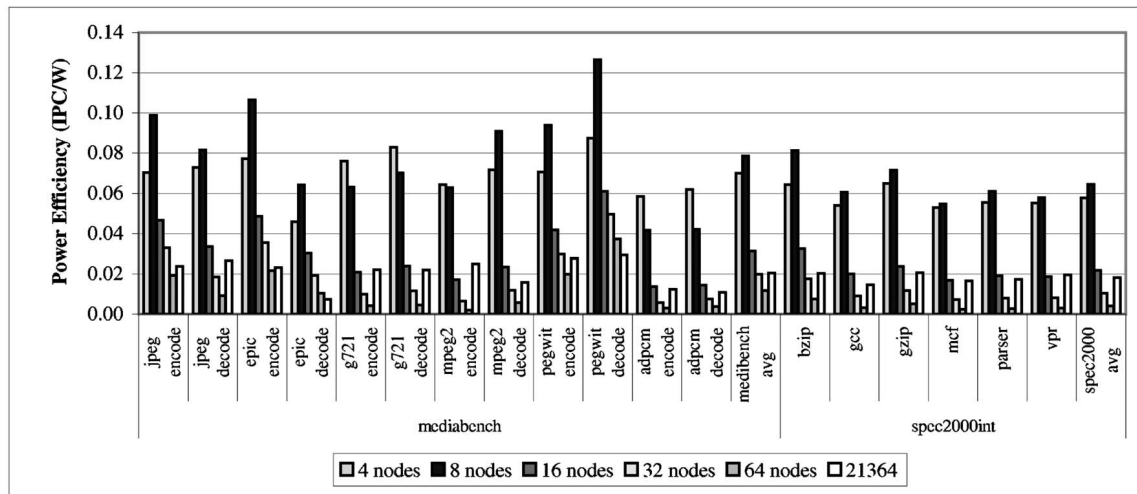


Fig. 5. Power efficiency (IPC per watt) for Atlas configurations and OOO baseline machine.

### 6.3 Silicon Efficiency Results

By dividing IPC by the projected area of the chip, we create an efficiency metric for justifying silicon area. GENESYS computes the area for the conventional OOO processor as  $150\text{mm}^2$ , well under the ITRS estimate for maximum processor size at 65nm. Fig. 4 shows these efficiency numbers for the same benchmarks and configurations as before. The smallest configuration (4-node) is clearly the most silicon efficient, though this trend holds true with conventional uniprocessors as well. The aggressive OOO processor shown, for instance, is designed for high integer performance at the expense of transistors and thus places below even the 16-node Atlas configuration. IPC drop-offs indicate that larger Atlas configurations cannot justify their area, but, as this design is mostly a replication of a simple verifiable core, equivalent Atlas transistors should be easier to verify than OOO transistors.

### 6.4 Power Efficiency Results

By dividing IPC by the GENESYS power estimate for each configuration, we compute a power efficiency metric. Fig. 5 shows these results for all configurations. GENESYS computes the power estimate for the uniprocessor baseline at 104W, well within ITRS guidelines for 65nm processors. The figure shows that Atlas power efficiency peaks at eightnodes, unlike area efficiency, which peaked at four nodes. After this point, however, efficiency drops predictably. The power efficiency of the OOO design is near that of the 16-nodes Atlas design, showing that CMPs with dynamic multithreading have potential for power savings over similar performance uniprocessors. Atlas power load can be further decreased by gating off entire nodes at energy-critical times.

### 6.5 Sensitivity

To test the sensitivity of our IPC results, two key assumed parameters are varied. First is the global communication latency, which was determined by the earlier layouts and the assumption of one cycle delay per grid cell in our timing analysis. We measure how the average performance of all applications would scale differently if this propagation time was doubled or tripled. These scenarios are presented in the left two scaling curves of Fig. 6. We also analyze changing the value/control predictor lookup from single cycle access. Scaling results for a three and five cycle predictor are shown in the right two curves of the figure. For reference, the center scaling curve is the normal case which was presented earlier, in Fig. 3. It is evident that performance is not highly dependent on value predictor lookup time. Performance drops by less than

15 percent in the far right curve, despite the predictor being five times slower to access. Communication latency, however, greatly affects performance. IPC drops by more than 30 percent when the delay to the reach the predictor and L2 are doubled. As execution is slowed by increased VP latency, the communication effect is dominated by the increased L2 round trip times.

## 7 CONCLUSION

Aggressive uniprocessors dominate the current processor market despite their difficult validation and their low area and power efficiency. These trends only worsen with each new generation, despite the demand for low-power and short design cycles. On the other hand, the scalable nature of Atlas's ring design allows applications, given decent control and value predictability, to achieve greater performance and efficiency than on a conventional OOO processor. To confirm scalability of such a design, we have proposed layout goals such as maintaining the data-forwarding ring between adjacent nodes and placing global structures in the center of this ring. These heuristics allow Atlas designs with dozens of nodes to be laid out simply. Though configurations up to 32 nodes are feasible given ITRS predictions for 65nm production, area and power efficiency (and often performance) beyond

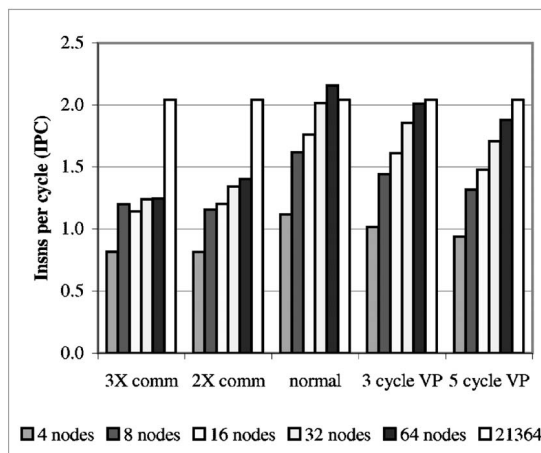


Fig. 6. Average IPC for all benchmarks when global latencies are doubled or tripled and when the predictor latency is three or five cycles.

16 nodes declines below that of a uniprocessor. Simulations with 4, 8, and 16 node Atlas configurations and accurate global latencies, however, show that these design points provide an efficient means of executing sequential binaries.

## REFERENCES

- [1] H. Akkary and M. Driscoll, "A Dynamic Multithreading Processor," *Proc. 31st Int'l Symp. Microarchitecture*, 1998.
- [2] S. Nugent, D. Wills, and J. Meindl, "A Hierarchical Block-Based Modeling Methodology for SoC in Genesys," *Proc. IEEE Int'l ASIC/SoC Conf.*, 2002.
- [3] E. Rotenberg, Q. Jacobson, Y. Sazeides, and J. Smith, "Trace Processors," *Proc. 30th Int'l Symp. Microarchitecture*, 1997.
- [4] G.S. Sohi, S. Breach, and T. Vijaykumar, "Multiscalar Processors," *Proc. 22nd Ann. Int'l Symp. Computer Architecture*, 1995.
- [5] S. Chappell, J. Stark, S. Kim, S. Reinhardt, and Y. Patt, "Simultaneous Subordinate Microthreading," *Proc. 26th Ann. Int'l Symp. Computer Architecture*, 1999.
- [6] B. Nayfeh and K. Olukotun, "A Single-Chip Multiprocessor," *Computer*, vol. 30, no. 9, Sept. 1997.
- [7] L. Codrescu, D. Wills, and J. Meindl, "Architecture of the Atlas Chip-Multiprocessor: Dynamically Parallelizing Irregular Applications," *IEEE Trans. Computers*, vol. 50, no. 1, pp. 67-82, Jan. 2001.
- [8] L. Codrescu and D. Wills, "On Dynamic Speculative Thread Partitioning and the Mem-Slicing Algorithm," *Proc. Seventh Ann. Int'l Conf. Parallel Architectures and Compilation Techniques*, 1999.
- [9] V. Sarkar and J. Hennessy, "Partitioning Parallel Programs for Macro-Dataflow," *Proc. 1986 ACM Conf. Lisp and Functional Programming*, 1986.
- [10] Z. Purser, K. Sundaramoorthy, and E. Rotenberg, "A Study of Slipstream Processors," *Proc. 33rd Int'l Symp. Microarchitecture*, 2000.
- [11] J. Steffan, C. Colohan, A. Zhai, and T. Mowry, "A Scalable Approach to Thread-Level Speculation," *Proc. 27th Int'l Symp. Computer Architecture*, 2000.
- [12] S. Wilton and N. Jouppi, "Cacti: An Enhanced Cache Access and Cycle Time Model," *IEEE J. Solid State Circuits*, vol. 31, no. 5, May 1996.
- [13] "Intel Microprocessor Quick Reference Guide," <http://www.intel.com/pressroom/kits/quickreffam.htm>, 2003.
- [14] I.T.R. for Semiconductors, "Executive Summary, 2003 Edition," <http://public.itrs.net/Files/2003ITRS/ExecSum2003.pdf>, 2004.
- [15] D. Burger and T. Austin, "The SimpleScalar Tool Set, version 2.0," Technical Report 1342, Computer Science Dept., Univ. of Wisconsin-Madison, 1997.
- [16] C. Lee, M. Potkonjak, and W. Mangione-Smith, "Mediabench: A Tool for Evaluating Multimedia and Communications Systems," *Proc. 30th Int'l Symp. Microarchitecture*, 1997.