

A Wire-Delay Scalable Microprocessor Architecture for High Performance Systems

S.W. Keckler⁺, D. Burger⁺, C.R. Moore⁺, R. Nagarajan⁺,
K. Sankaralingam⁺, V. Agarwal^{*}, M.S. Hrishikesh^{*},
N. Ranganathan⁺, and P. Shivakumar⁺

⁺Department of Computer Sciences

^{*}Department of Electrical and Computer Engineering
The University of Texas at Austin, Austin TX

While microprocessor pipeline depth has increased dramatically over the last decade, such deep pipelining is close to a hard limit. As shown in Figure 1, the number of logic levels in modern processors is approaching that of the Cray 1-S, measuring approximately 8 fanout-of-4 (FO4) inverter delays. Due to constraints of pipeline overheads and power consumption, continuation of this trend will be prohibitively difficult[1]. Furthermore, global on-chip wire delays will grow significantly, increasing cross-chip communication latencies to tens of cycles and rendering the expected chip area reachable in a single cycle to be less than 1% in a 35nm technology, as shown in Figure 2. The challenge then is to design new architectures that enable a very low FO4 design, while compensating for the decrease in pipeline scaling by increasing concurrency, despite slow global wires. Because existing superscalar microarchitectures rely on global communication, they are poorly matched to the technology challenges of the coming decade.

The Grid Processor architecture (GPA) is designed to address these technology challenges[2]. As shown in Figure 3, each GPA implementation consists of a 2-D array (here 4x4) of ALUs connected via a routed operand network, with L1 I-cache, D-cache, and register file banks around the periphery of the ALU array. Each ALU includes an integer unit, a floating point unit, instruction buffers, operand buffers, and an operand router. While the ALU chaining is similar to that proposed in [3] and the partitioning strategy is akin to clustered VLIW architectures, the GPA approach permits out-of-order execution and fast clock rates, enabling performance far higher than conventional architectures.

In a Grid Processor program, spatial instruction scheduling is static but execution order is dynamic. The compiler forms large single-entry, multiple exit regions (hyperblocks) and schedules them to the ALU array. Current hyperblock generation techniques yield instruction blocks consisting of 15-120 (average of 48) useful instructions, each with typically fewer than 10 input and 10 output registers, for SPECINT-2000 benchmarks. The instructions along the critical path of the block are mapped to the shortest possible physical path in the grid, as shown in Figure 4. At runtime, the instructions of a block are fetched en masse from the multiported instruction cache and distributed horizontally into the grid. Instructions execute in dataflow order, dictated by the arrival time of the operands at each ALU. Intermediate values are routed directly from the producing ALU to the consuming ALU without being written back to the register file or being broadcast throughout the grid. Block outputs are written to the register file at block termination and are bypassed directly to instructions in the next block. Conditional move instructions ensure that the block outputs produced correspond to the correct block exit point. A next-block predictor speculatively selects a subsequent block to be mapped and executed while the current block is being executed. Mis-speculations and exceptions cause rollback to the last committed block boundary. Figure 5 shows the effect of these features on attainable instruction-level parallelism (ILP), comparing an 8x8 GPA to the Alpha 21264.

The GPA offers inherent technology scaling advantages over conventional architectures. First, it encourages the use of natural partitioning in the ALU array, the banked caches, and the banked register files, providing both faster access and higher bandwidth to the memory structures. Second, communication delays in the ALU array are exposed to the compiler for optimization, reducing the need for broad communication networks within core. Third, GPAs enable block-oriented state tracking and orchestration, as opposed to the conventional instruction-oriented approaches. This block atomic model serves to eliminate many per-instruction overheads and centralized structures associated with instruction fetch, register read, commit, and exception handling. Finally, the instruction buffers associated with each ALU serve as a set of distributed reservation stations enabling an effective dynamic scheduling window of hundreds or thousands of instructions.

GPA machines provide unique opportunities for power efficiency. The elimination of structures dedicated to instruction-level register renaming, associative operand comparisons, and state tracking effectively reduce the overhead circuitry and power on a per ALU basis. In addition, ALU chaining serves to dramatically reduce the number of global register file accesses in exchange for short point-to-point connections. The dynamic power of the ALU array and banked memory structures can be actively managed to reduce consumption during periods of lighter utilization. The dataflow execution model of the GPA is also amenable to power efficient asynchronous design techniques.

In addition to high ILP, a secondary design goal of the GPA is *polymorphism*, or the ability to adapt the hardware to the execution characteristics of the application. Grid Processors can be easily subdivided into smaller sub-processors, allowing discrete threads to be assigned to different sub-processors for high thread-level parallelism (TLP). Grid Processors can also be configured to target data-level parallelism (DLP), which is often exhibited in media, streaming, and scientific codes. For these applications, the grid is subdivided into physical *row processors*, each with its own I-cache bank, row of processors, and D-cache bank. Instructions for kernels or inner loops are mapped to the processors and stay resident for multiple iterations. Accesses to the data cache banks provide multiple values that are distributed to the ALUs in its row. Initial results show that a GPA can average up to 48 compute instructions per cycle on an 8x8 grid. Assuming an 8 GHz clock in 50nm CMOS, this configuration would provide performance of 384 GFlops.

As shown in Figure 6, we are planning prototype chips that will consist of a small number of powerful 16-ALU cores, a shared L2 cache structure built from an array of 128KB memory banks connected by a routed network, a set of distributed memory controllers, and a set of channels to external memory. The prototype will be built using a 130nm process and is targeted for completion in 2005.

References

- [1] M.S. Hrishikesh, N.P. Jouppi, K.I. Farkas, D. Burger, S.W. Keckler, and P. Shivakumar, "The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays," *29th International Symposium on Computer Architecture*, pp. 14-24, May, 2002.
- [2] R. Nagarajan, K. Sankaralingam, D. Burger, and S.W. Keckler, "A Design Space Evaluation of Grid Processor Architectures," *34th Annual International Symposium on Microarchitecture*, pp. 40-51, December, 2001.
- [3] M. Ozawa, M. Imai, Y. Ueno, H. Nakamura, and T. Nanya, "Performance Evaluation of Cascade ALU Architecture for Asynchronous Super-Scalar Processors," *International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 162-172, March, 2001.

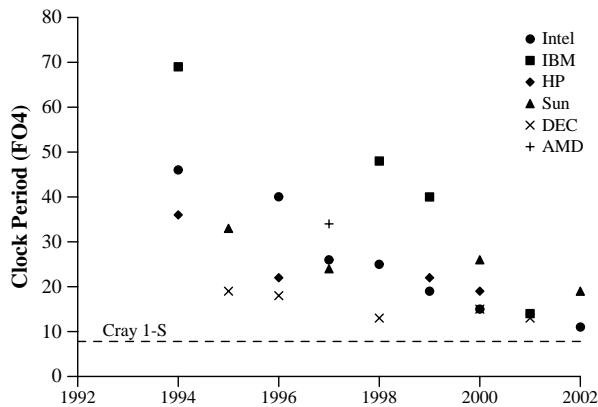


Figure 1: Historical reduction in cycle time driven by pipelining.

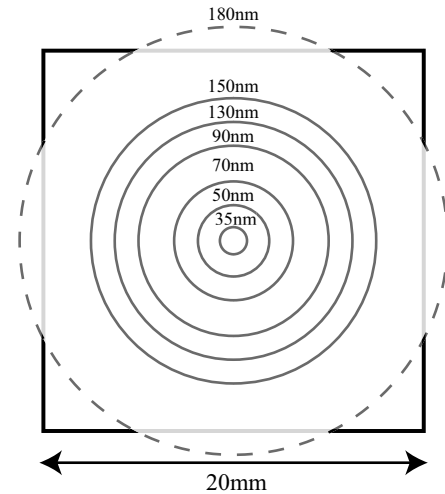


Figure 2: Projected fraction of chip reachable in one cycle.

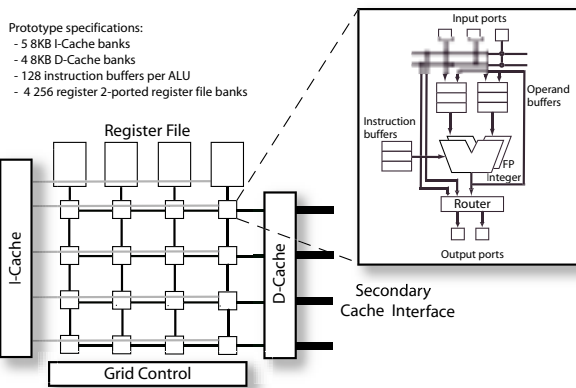


Figure 3: Grid Processor block diagram.

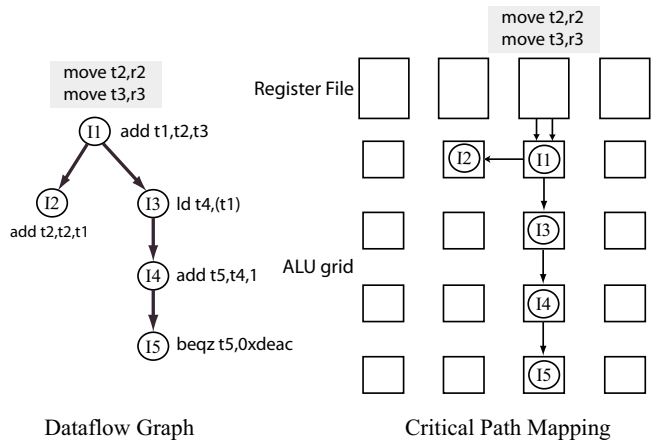


Figure 4: Mapping of dataflow critical path to physical ALUs.

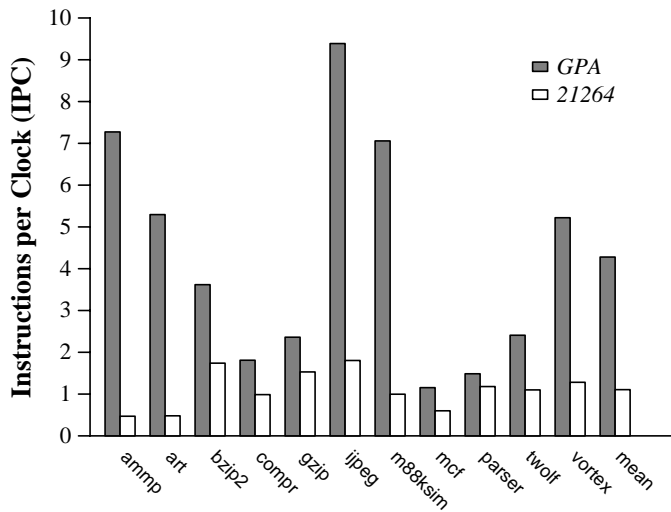


Figure 5: GPA achieves 1.3-15x greater instructions per clock (IPC) than conventional out-of-order core.

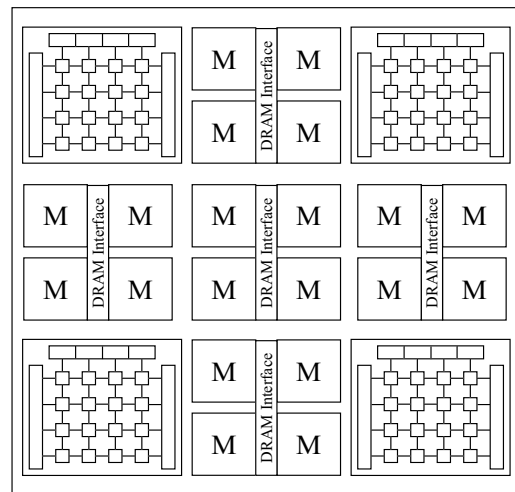


Figure 6: Diagram of expected prototype of chip-multiprocessor (CMP) composed of GPAs.