POD: A 3D-Integrated Broad-Purpose Acceleration Layer

To build a future many-core processor, industry must address the challenges of energy consumption and performance scalability. A 3D-integrated broad-purpose accelerator architecture called parallel-on-demand (POD) integrates a specialized SIMD-based die layer on top of a CISC superscalar processor to accelerate a variety of data-parallel applications. It also maintains binary compatibility and facilitates extensibility by virtualizing the acceleration capability.

Dong Hyuk Woo
Hsien-Hsin S. Lee
Georgia Institute of Technology

Joshua B. Fryman
Allan D. Knies
Marsha Eng
Intel

With the continuing trend of feature-size scaling and process technology advancement, integrating 10 to 100 billion transistors on a reasonable die area will likely become feasible by 2015.[1] Instead of continuing to enlarge on-die cache capacity, the trend is to improve performance and throughput by exploiting thread-level parallelism with a massive number of processor cores on die.[1,2] Currently, most general-purpose multicore designs leverage off-the-shelf processor economies of scale and simply adopt a symmetric-multiprocessing (SMP) style many-core architecture. Nevertheless, to integrate hundreds or even thousands of cores on a single die, the fundamental physical limit, power consumption must be addressed to make such a many-core architecture viable. For future many-core processors, low power will be not just a feature, but a design constraint. Achieving the goal of integrating a large number of cores onto one chip boils down to one issue: how to maintain power efficiency. In other words, how can architects better arrange these resources to maintain performance scalability without exceeding a given power envelope from both power-supply and thermal-management perspectives?

One school of thought is to incorporate on-die special-purpose accelerators with general-purpose cores. From an area-efficiency standpoint, however, it's impractical to specialize and accelerate all possible applications of interest. Furthermore, implementing several application-specific accelerators on a general-purpose platform has many drawbacks, such as longer design turnaround time, higher nonrecurring engineering cost, inflexibility, and lack of backward or forward binary compatibility.

To address these issues, we introduce a broad-purpose accelerator design called *parallel-on-demand* (POD). Based on CISC superscalar processors (such as the Intel 64, formerly known as the Intel EM64T, processor), POD revisits several massively parallel SIMD designs yet focuses on novel challenges including backward and forward binary compatibility issues, on-chip wire delay, and efficient interaction with a superscalar host processor. Furthermore, using

Published by the IEEE Computer Society

emerging 3D die-stacking technology, a processor can flexibly snap a POD layer on top of a conventional general-purpose processor die with the ability to virtualize different generations of a POD design. POD's snap-on feature also lets processor vendors optionally upgrade a product during the packaging phase.

POD architecture

Figure 1 is a high-level block diagram of the POD architecture, which consists of two die layers: a general-purpose processor (an Intel 64 processor) as the bottom layer and a snap-on accelerator, the POD layer. The general-purpose processor layer can be a die of a conventional multicore processor with design hooks. Depending on the target market segment, we can stack a POD layer on top at packaging time to improve performance per joule for certain applications such as high-definition multimedia, 3D games, or scientific computing. (The *performance* in performance per joule is defined as the inverse of execution time.)

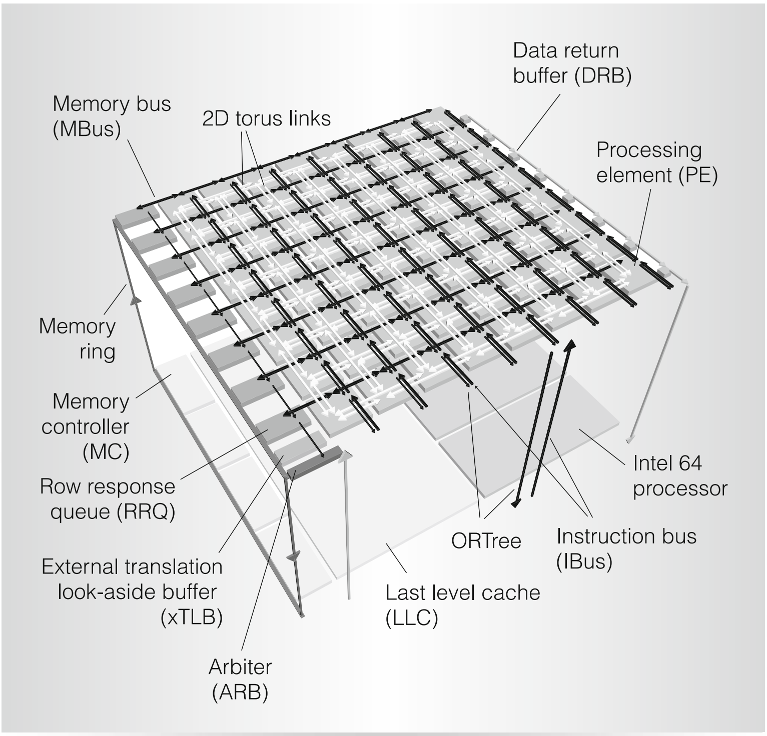

Figure 1. POD architecture in stacked die arrangement.

During operation, the host Intel 64 processor fully boots a normal OS and runs every legacy application under that OS without deviating from current performance. At the instruction decode stage, the host processor might encounter a block of code written for POD to accelerate. In such a case, instructions within the block are broadcast to the POD layer through an instruction bus (IBus), of which each instruction has the same fixed size. The processor might also broadcast 64-bit immediate values to the POD's registers.

The POD layer is essentially a massively parallel SIMD processing element (PE) array. The target SIMD PE array is a sea of $n \times n$ tiles, where $n = 8$ in Figure 1. The PE array executes instructions broadcast through the IBus and generates a flag-tree output that is tied together logically via an OR gate (ORTree) and is routed back to the host pipeline. The computed results from the PEs can be retrieved through the data return buffer (DRB) or memory hierarchy.

Heterogeneous ISA

Each PE tile contains a high-performance arithmetic unit with its own private register file and local SRAM memory (see Figure 2). To provide a baseline performance level and support a subset of the host instruction set to facilitate PE virtualization, we use an existing 128-bit streaming SIMD extension (SSE) engine from a contemporary Intel 64 processor in POD.

Although each PE will ideally be capable of decoding conventional CISC instructions for execution, this approach will require a large instruction decoder for each PE. This is less desirable when considering performance per square millimeter or performance per joule. Instead, to make each PE compact and efficient, we chose a VLIW execution model for the PE. The PE instructions, each 12 bytes wide, are broadcast from the host processor. Each 12-byte word forms a partially predecoded VLIW packet of three instructions that eliminate the CISC decoding overheads in POD. Each VLIW packet has a fixed format of one G (generic), one X (SSE), and one M (memory) pipeline instruction. Because the host processor orchestrates the execution of PE instructions, there's no need for imple-

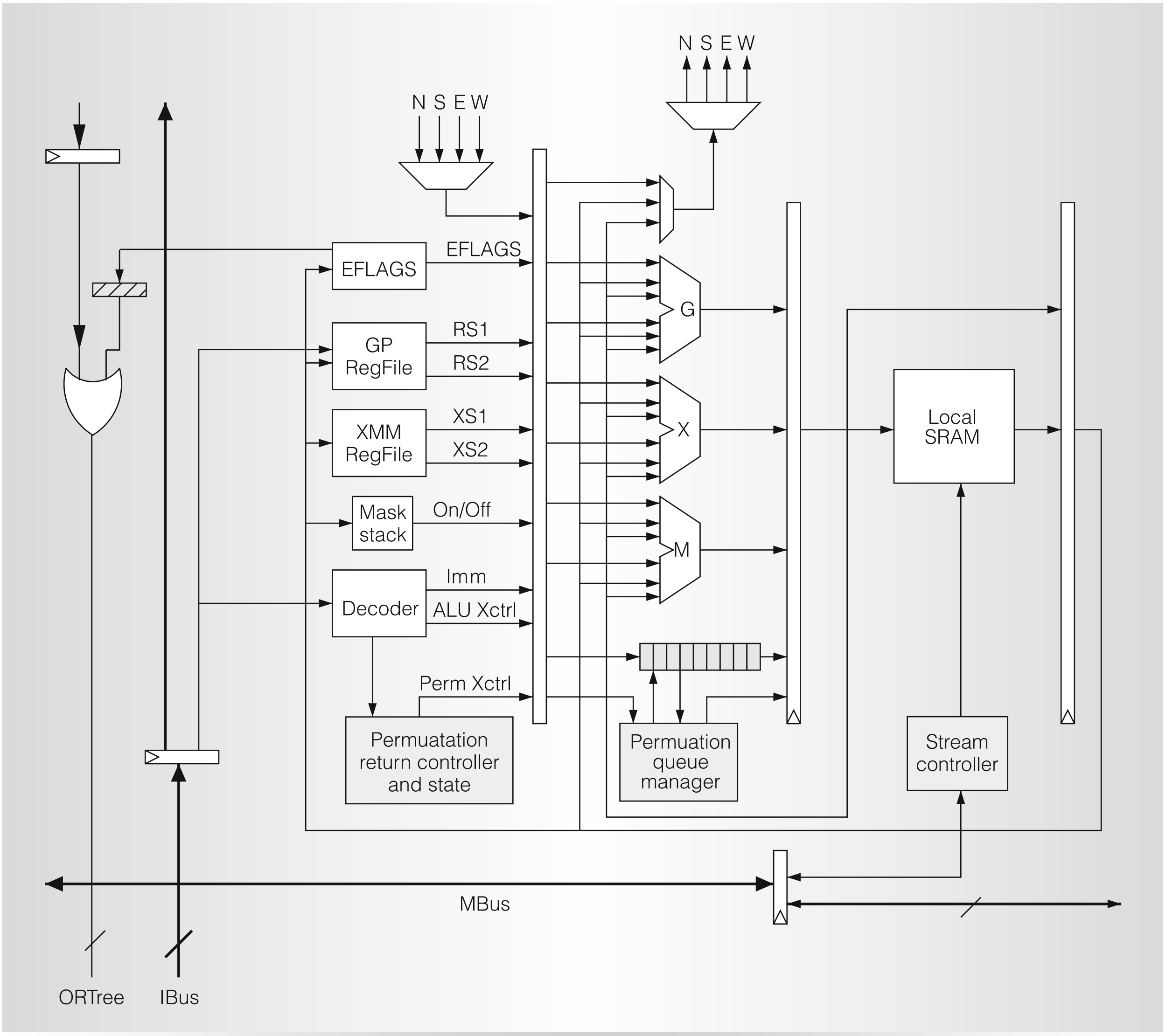

Figure 2. A processing element tile.

menting an instruction cache within the PE. Furthermore, because each PE is executing the same instruction and there's no instruction equivalent to a branch, no branch predictor or associated flush/control logic is required, keeping the PE small and simple.

Because the host processor will orchestrate and broadcast POD instructions, we enhanced the Intel 64 instruction set architecture (ISA) to handle such heterogeneous instructions. To enable this, we added a new instruction prefix byte called *SendBits* indicating a POD instruction to the existing Intel 64 ISA. When this prefix byte is encountered, it indicates the following 12-byte word is an encapsulation of three POD operations. The host processor will then dispatch the 12-byte VLIW to the POD array. Also, we rely on the compiler or assembler to schedule and pack the three-way VLIW instructions.

Unlike conventional massive SIMD machines, the POD integrates a massive SIMD PE array with a modern out-of-order host processor. To ensure execution correctness, two major challenges must be addressed in the host processor: recovery from misspeculation and out-of-order dispatch of POD instructions. To support speculative execution, some recovery mechanism is required

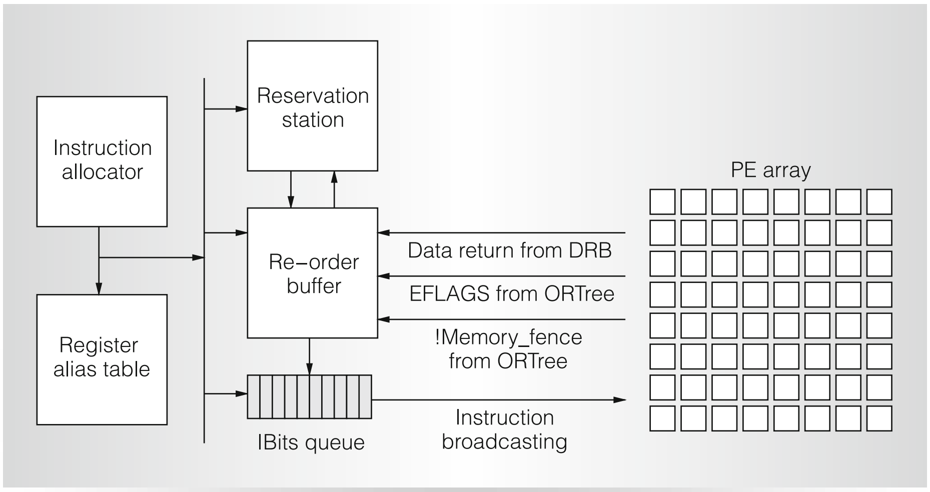

Figure 3. IBits queue in the superscalar pipeline.

to roll the machine back to the correct architectural state. For example, Tarantula, which had relatively narrow SIMD engines attached to a superscalar processor, implemented a recovery mechanism in each PE.[3] Unfortunately, this results in substantial overhead to both the area and power for a massively parallel SIMD engine. To simplify the PE design (and build as many PEs as possible), our host processor is designed to broadcast POD instructions in a nonspeculative manner. In other words, the POD instructions won't be dispatched from the host processor until its preceding branches are resolved. From a performance standpoint, as long as the code that runs on the host processor doesn't depend on the results from the POD, this approach won't degrade performance. (This is the case for all of our benchmark programs we evaluated except k-means. The host processor doesn't issue any data-dependent instruction that reads data updated by the POD immediately for these benchmark programs. This event is extremely rare even in k-means simulation.)

Another issue is that the host processor might reorder the POD instructions, which might lead to incorrectness because the PE is ignorant of program order. To prevent this, the POD instructions the host processor issues are strongly ordered by implementing an IBits queue along with a conventional out-of-order pipeline (see Figure 3), similar to the store queue found in an out-of-order processor. When a SendBits instruction is issued, its 12-byte immediate field (encoding a VLIW POD instruction) is entered into the IBits queue. Upon the retirement of the SendBits instruction from the reorder buffer (ROB), the corresponding 12-byte immediate value is latched onto the IBus and is broadcast to PEs.

Wire-delay-aware design

To enable SIMD-style instruction execution where every PE executes each instruction at the same global clock cycle, there are two options:

- executing an instruction immediately upon arrival to a POD row, leading to a north-south time-zone effect, or
- buffering each arriving instruction for sufficient time such that every PE will execute the same instruction at the same instant.

The time-zone effect can be challenging for programmers and architects to work around, as any given row will be executing instruction j, while the preceding row is executing $j + 1$ and the successor row is executing $j - 1$. To avoid undesired complexity for programmers, architects, and compilers, we use a buffering model to eliminate the time-zone effect and enable lock-step execution.

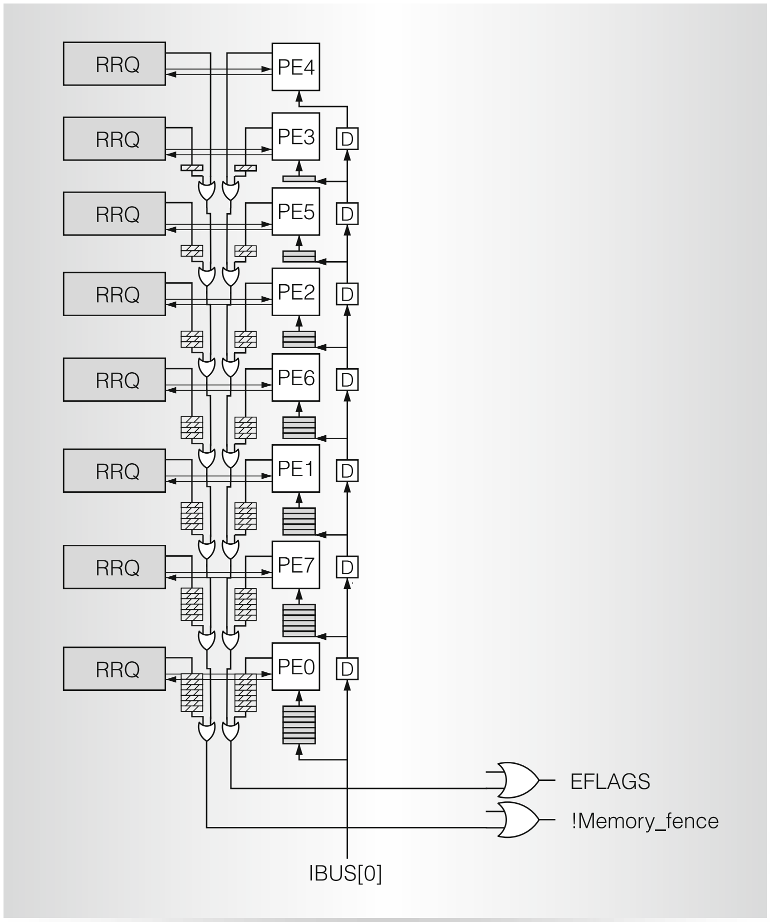

Figure 4. A detailed POD column.

Figure 4 shows such a model for a single column in the POD. Instructions are broadcast using the IBus and are queued before being executed by the PE. For n rows, it takes $n - 1$ cycles before every PE executes the instruction. The queue size shrinks monotonically as a PE's location gets farther away from the host processor. (Although each PE is identical, programmable fuses are burned after die manufacture to set the used numbers of slots in the queue.) For an $n \times n$ POD, where $n = 8$, there are seven entries for the bottom-most PE, while no queue is needed for the topmost PE. The delay units (D block) are inserted to delay each instruction broadcast to synchronize the SIMD execution. Similarly, when gathering results (such as EFLAGS) from PEs, the results from the PEs closer to the host processor must be delayed and wait in their queue until the farther results arrive for combining. Figure 4 depicts these in the propagation paths with correct delay queues on the left side. Compared to the previous immediate execution model, there's no overhead to the PEs with this implementation, except for a buffer to hold the instructions broadcast. The roundtrip latency for the host processor to evaluate the conditional loop also remains the same, which for n rows, is $2 \times (n + f_0) + l$ cycles, where $2n$ cycles are consumed for instruction and EFLAGS propagation, $2f_0$ is fan-in and fan-out latency between the host processor and PEs in the first row, and l is the actual instruction latency.

Regardless of how the PEs are connected to each other, the instruction and data-value broadcast from the host processor are arranged as a fan-out tree. The PE array's bottom-most row will receive the same instruction at the same cycle. The instruction is then passed up to the next row in sequence. This fan-out tree is implemented on a POD layer so that the number of die-to-die vias between a general-purpose processor layer and a POD layer remains unchanged (Figure 1), regardless of the number of PEs. This constant interface lets us extend a POD layer without redesigning a general-purpose layer every time we introduce a new product.

Energy-efficient interconnection network

Figure 1 also shows that POD adopts a folded torus network.[4] To minimize latency and maximize packing, each PE is small enough that signal propagation time over one PE is less than one clock cycle. Ideally, each side of a PE will be no longer in any direction than 95 percent of the wire distance in one clock cycle with all surrounding line drivers, buffers, and so forth. The ordering of the number labels inside the PEs of the leftmost column in Figure 4 indicates the north-south nearest-neighbor connection pattern. In the same way, the communication links for each row are laid out in east-west direction. In addition to providing shorter links, such a layout also leads to deterministic communication latency. The significance of this is that we can disable communication-related logic and wires safely when they are not used. Moreover, the lock-step execution model will make the entire computation predictable and thus fully debuggable. There are no tricky issues such as race condition, live lock, deadlock, and so on found in normal SMP-style many-core processors.

At any given moment, only one direction (input and output) must be enabled. Because each nearest-neighbor communication pattern has a known latency and is directed by communication instructions, the links are disabled during the course of pure computation or when links in the other direction aren't being used. Furthermore, no power-hungry routers are required for this point-to-point network. All we need are a single 4:1 multiplexer and a 1:4 demultiplexer for input and output. With this reduced power profile, the POD array's growth is only limited by the average power consumption of each PE and the manufacturing die reticle. This approach contrasts with other tiled designs such as the Raw processor from MIT or the TRIPS processor from the University of Texas where any of the interconnection network links could be active at the same time due to dynamic routing.

Each PE can communicate with its nearest neighbor by either directly moving a register value of up to 128 bits or by transferring memory in 64-bit chunks. Because the nearest-neighbor latency for a folded torus is targeted to be two cycles or less, this allows for high-throughput computation even when the algorithm requires neighboring registers and memory values. A fully synchronized computation model allows register transfer operations to utilize full communication bandwidth without any network overhead, such as additional latency due to contention, and header encoding overhead. In the case of memory transfer operations, only a half communication bandwidth can be utilized because the upper 64 bits are used to encode other information such as memory addresses.

When one PE needs to communicate to another PE in a non-nearest-neighbor fashion, we use the k-permutation routing in our interconnect design.[5] Rather than providing dynamic-wormhole-routing hardware support for a relatively infrequent operation, we use a dedicated algorithm to drive the collective POD multiplexers into a series of sweeps to migrate all data to the intended targets. These algorithms require each PE to support n hardware buffer slots (permutation queue) of the bit size matching the point-to-point link width in an $n \times n$ SIMD array.

The basic algorithm proceeds by having all PEs send messages to the east, with each message stopping when it reaches its target column. This takes $n - 1$ hops, and at the end at most n messages will be buffered in any one PE. At the end of this sweep, every message in every POD row will be in its target column. If we now apply the same algorithm to the north, we may require as many as n^2 steps until all the buffered messages reach their target PE. As messages reach their target PE, they are processed (stored into the appropriate memory location). This two-phase sweeping algorithm ensures that for any permutation of routing, even all-to-one, all messages are delivered after a fixed latency. This fixed routing wouldn't be an optimal solution, but each PE needs to enable only one link at the same time, which is more energy efficient. Although the fixed latency might be high for such generic routing support, we've made the trade-off to keep nearest-neighbor communications fast, which is a much more frequent event than generic routing. More optimized row-only and column-only sweeps of just $n - 1$ steps are also supported for more structured communication to reduce the high latency of a full any-to-any communication.

Virtual address support

Aside from a 128-Kbyte private local SRAM dedicated to each PE, applications must also be able to communicate with the system memory through normal loads and stores. To manage this interaction, each PE is further enhanced with two unidirectional memory buses (MBuses) to the main memory via an interface called the row response queue (RRQ). One bus streams data back from main memory to the PEs in the row, while the other bus streams data from the PEs in the row to the main memory. Because the system memory operations of all PEs are synchronized by barrier operations, PEs can safely disable their MBus and its related logic to minimize energy consumption when they aren't communicating with the system memory. The RRQ is the queuing point for transactions in both directions and, in turn, is connected to a memory ring with the host processor's last level cache (LLC) and all memory controllers (MCs). The ring is good not only for easy arbitration and high throughput, but also for a POD layer and a general-purpose layer to be merged into one system, as Figure 1 shows.

System memory accesses use virtual addresses acquired from the host processor. All n^2 PEs share one pipelined translation look-aside buffer (TLB), which is external to the host processor, but managed by it. The external TLB (xTLB) in Figure 1 need not be organized along traditional lines; the TLB lookup isn't as critical as it is in the host processor. This allows for a super-pipelined, high capacity xTLB to be implemented, much like the texture sampler TLB in a general-purpose graphics processing unit. In the event of a fault or miss event in the TLB, the host processor is notified and the request in the RRQ control ring is flagged as a TLB failure. When the host processor updates any TLB entry, a dedicated control signal in the RRQ control ring is set to indicate that any prior TLB failure may now retry.

Minimal ISA modification in host processor

As we discussed earlier, the host processor manages the SIMD execution inside PEs completely. To enable this, the host processor ISA is extended with five new instructions and three modified instructions. The new instructions are

- SendBits, to broadcast instructions to PEs;
- GetFlags, to obtain the return status;
- DrainFlags, which assures that the initial setup of a known state in the flag tree is complete;
- SendRegister, to broadcast a host register value to PEs; and
- GetResult, to obtain a return buffer value from PEs without using system memory as a go-between.

The three modified host instructions are the various fence operations (load, store, and combined) that are extended to monitor the return status of the POD's memory interface system.

POD virtualization

There are two main reasons for virtualizing the POD accelerator. The first is to provide execution compatibility for various POD sizes. Without such resilience in the design, software vendors would need to recompile their code to fully utilize all PEs for each particular platform. To virtualize the number of PEs, we hardwired six variables in POD: the number of PEs in each row, the number of PEs in each column, the number of PEs, each PE's *x*- and *y*-coordinates, and the PE ID. The host processor can retrieve the first three values by using a CPU identification (CPUID) instruction, and each PE can retrieve all six values from the protected memory space of its local memory, which is preset at boot time. By forcing each PE to read these six values from its local memory, the same POD binary code can continue to improve performance as the number of PEs grows.

The second reason for POD virtualization is to circumvent the compatibility issue of running POD code on a platform without an integrated POD acceleration layer. There are several potential solutions; one can rely on a software or hardware binary translator. Because the POD ISA inherently originates from Intel 64 ISA, the POD code can be dynamically translated into Intel 64 ISA-compatible instructions. Three-operand POD operations can be translated into two-operand Intel 64 ISA instructions at the cost of less efficiency. (An fma, or floating-point multiply and add, instruction needs to be translated into two Intel 64 ISA instructions, and it introduces a rounding error between the original code and the translated code.) The fact that the number of PE registers is greater than that of the host processor can result in reduced efficiency. Communication instructions can be simply ignored as if there is only one PE. Masking operations can be converted into branch instructions. Local memory of a PE can be emulated by copying the original data into a virtual memory space of the same size as the local memory of a PE.

Another solution is to implement one PE on the general-purpose processor layer to execute the POD code natively. The original goal of a PE design was to maximize area, power, and communication efficiency by minimizing a PE. This minimal design, when integrated onto the host processor die, increases the area of the host processor layer by 9 percent (not considering the area of the L2 cache) based on our area analysis but can provide better performance than the software translation or hardware translation approach. The 9 percent area penalty could be further subsidized by reusing the existing SSE unit in the host processor.

Physical design evaluation

POD aims for a 3-GHz clock speed assuming a 45-nm or better process. For this target frequency, the memory ring is capable of a bandwidth up to 192 gigabytes per second (Gbytes/s), servicing up to eight 24-Gbytes/s MCs before any modification is required. For an 8×8 POD array, with each PE containing 128 Kbytes of SRAM, connected in a torus, the peak performance of single-precision and double-precision IEEE FP operations is 1.5 Tflops and 768 Gflops, respectively.

Based on published data from Intel (http://download.intel.com/technology/silicon/Bohr_IDF_Moscow_0406.pdf) and the die photo of its 45-nm Intel Xeon processor E5472 (formerly code-named Penryn), a single PE is estimated to occupy approximately 1.90 mm^2. In other words, the entire 8×8 PE array will amount to 122 mm^2. We assumed each RRQ, given the complexities of the various bus wirings and the ring interfaces, will be allotted an area on par with that of each PE. Therefore, the POD layer will amount to 137 mm^2. Compared to one E5472 core (22.26 mm^2), one PE consumes 9 percent of die area due to the lack of a CISC decoder, an instruction cache, branch predictors, TLBs, out-of-order execution related circuits, and so on.

Given that a PE consumes roughly 9 percent of die area of a single core, we used the E5472 product specification (http://processorfinder.intel.com/details.aspx?sSpec=SLANR) and a simple heuristic that power consumption of a certain block is proportional to the number of transistors in the block, to calculate that a PE will consume 1.37 W. Consequently, we expect 64 PEs and eight RRQs to consume 103.6 W. We additionally modeled the global interconnection power consumption using the Berkeley Predictive Technology Model.[6] We used 1.25 V, 3 GHz, and 0.5 for supply voltage, clock frequency, and switching factor, respectively. Based on these models, we expect the 96-bit IBus and two 80-bit MBuses to consume 1.90 and 3.16 W overall. This interconnect power is low mainly because all global communication links are highly pipelined, and each PE is extremely small.

In sum, we expect a POD layer to peak at 108.7 W. Assuming that a dual-core processor is bonded with a POD layer, and these two cores are fully utilized, we expect both layers to consume 148.7 W. In common scenarios, this maximum power is unlikely to be reached because the host processor won't be fully active while the POD layer is in operation. During this period, the host processor is only active for decoding encapsulated POD instructions, resolving branches for POD control, and so forth.

Performance evaluation

We developed a cycle-level POD simulator to carry out our performance study. This simulator models every single feature of the PEs and memory subsystem, including RRQ, xTLB, and memory controller. It also accurately models on- and off-chip communication bandwidth. In our simulation, off-chip DRAM bandwidth is modeled as 4×32 Gbytes/s (four on-chip memory controllers where each can provide 32-Gbytes/s bandwidth) and DRAM latency as 50 ns. To factor out performance improvement due to larger on-chip memory as the number of PEs increases, we assumed that the aggregate size of the on-chip memory remains the same regardless of

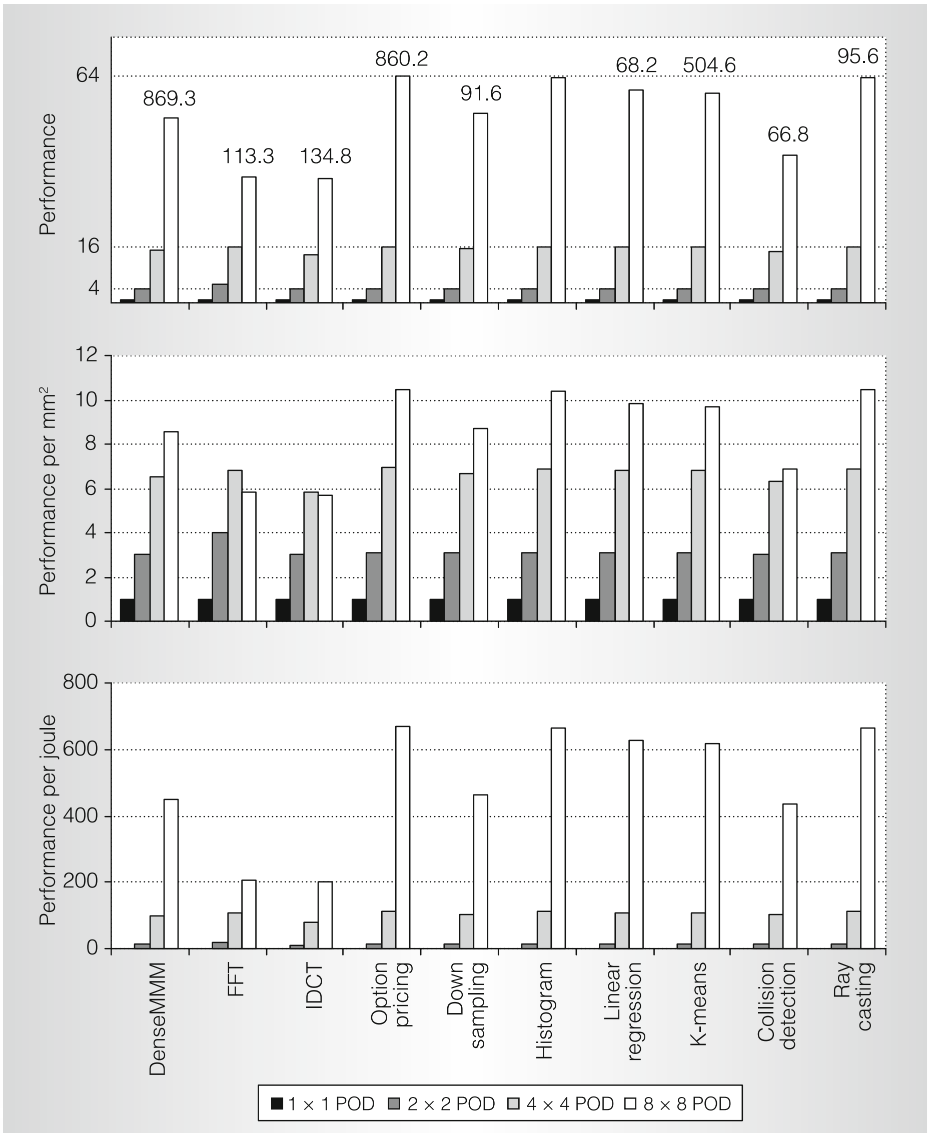

Figure 5. Simulation results. The bar numbers in the top graph are in absolute gigaflops. (Histogram: integer application, IDCT: double-precision floating-point application, Others: single-precision floating-point application)

the number of PEs for a fair comparison. For example, a PE of 1 × 1 POD has 8 Mbytes of local SRAM, while each PE of 8 × 8 POD has a 128-Kbyte SRAM only. We also conservatively assumed that the access latency of 8-Mbyte SRAM is equivalent to that of a 128-Kbyte SRAM, which is three cycles for a load or one cycle for a store. In reality, the access time of an 8-Mbyte SRAM of the baseline, a 1 × 1 POD, will be much longer, so the actual speedup will be even greater than our results indicate.

Figure 5 shows relative performance improvement, normalized to the performance result of a 1 × 1 POD. As the number of PEs increases, so does the achieved gigaflops. (We count each add, sub, mul, div, max, min, and cmp as one floating-point operation, and fma as two.) The figure shows that the trend of the speedup is approaching linear. When it runs

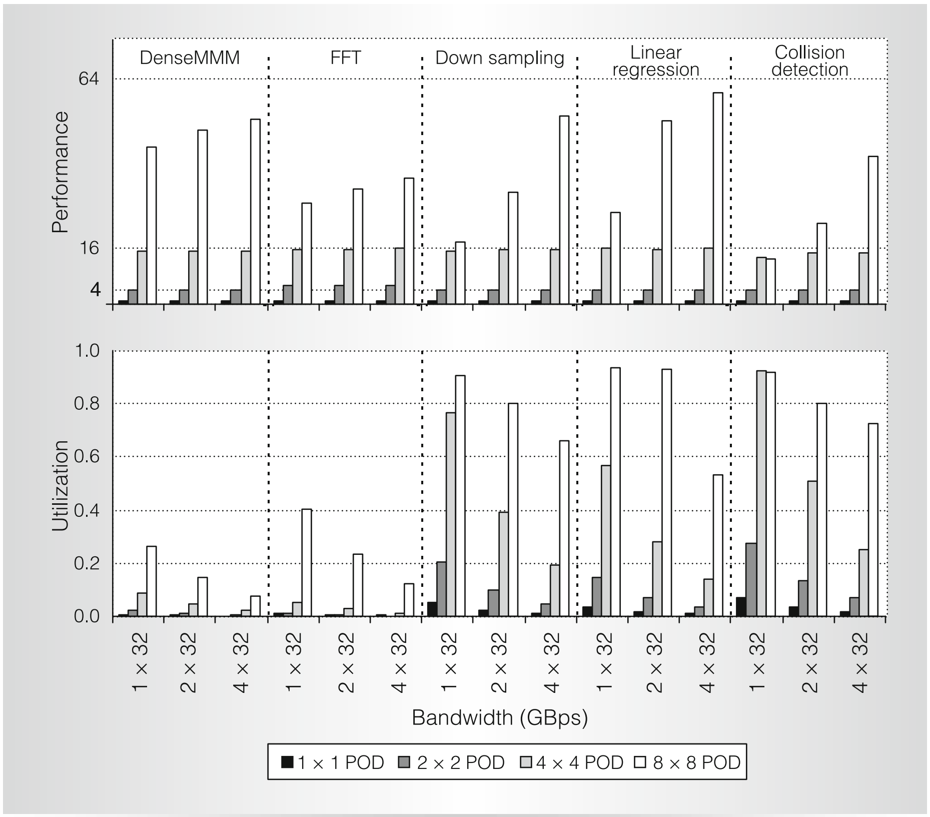

Figure 6. Effect of off-chip memory bandwidth.

a compute-intensive application, such as DenseMMM (dense matrix-matrix multiply) or OptionPricing (financial option modeling), it achieves more than 800 Gflops. In the case of DenseMMM, very low latency (two cycles) of neighbor-to-neighbor communication makes it possible to completely hide communication overhead with computation. The reason the performance doesn't show an ideal linear speedup is that the efficiency of each PE decreases, although not severely, because the working set of each PE becomes smaller when we increase the number of PEs to 64.

We found that the highly communication-intensive benchmark fast Fourier transform (FFT) also showed fairly good speedup. To demonstrate how effectively PEs exchange data, we chose a small input size (1,024 points) so that the computation latency couldn't hide the communication latency. Clearly, as the number of PEs increases, communication overhead becomes dominant, but we can still achieve good performance improvement because of our high-efficiency communication architecture.

An important observation is that, although a target application can be easily ported to POD for acceleration and its performance can be improved well, off-chip memory bandwidth can still be the greatest performance bottleneck in the future many-core era. Figure 6 shows the simulation results of five memory bandwidth-sensitive applications. The figure shows the performance improvement of different POD configurations with different off-chip memory bandwidth. Here, $x \times 32$ Gbytes/s means that the system has x on-chip memory controllers and channels, and each memory controller can support up to 32 Gbytes/s. As Figure 6 shows, memory bandwidth can be a serious bottleneck with on-chip many-core architectures when the number of cores is large. In these simulations, larger memory bandwidth improves an 8×8 POD's performance rather substantially in several benchmark programs. Once memory bandwidth is saturated, overall performance doesn't scale or can even degrade, as we can observe from the simulation result of the CollisionDetection (a physics simulation algorithm) with 1 × 32 Gbytes/s off-chip memory bandwidth. When the performance doesn't scale well due to saturated off-chip memory bandwidth, an intelligent mechanism to detect it and to disable some of the PEs to balance computation power and memory bandwidth would be an improvement.

To demonstrate the efficiency of POD as a snap-on accelerator, we used the following metrics for evaluation: performance per square millimeters and performance per joule. Figure 5 shows the relative area efficiency of POD, represented by performance per square millimeters. Because one PE consumes approximately 9 percent of the host processor's area, performance per square millimeters will keep improving as the number of PEs increases, given the overall performance scales. In an SMP-style many-core processor, this metric is at most one, because *n* cores consume *n* times space, and it can achieve *n* times speedup at the best scenario of linear speedup.

The performance per joule metric represents achievable speedup given a fixed energy budget such as battery lifetime and is equivalent to a reciprocal of energy-delay product.[7] Figure 5 shows that we can easily improve performance per joule with parallelization. All cases show that the larger the PE array, the more improvement this metric can achieve. Furthermore, performance per joule scales super-linearly. In the case of an SMP-style many-core system, execution time can be reduced by a factor of *n* times at most, while it consumes *n* times more power. Thus, the performance per joule of an SMP-style many-core system scales linearly in the best scenario of linear speedup. The additional performance per joule benefit of POD comes from the efficient design of PEs compared to the host processor.

In addition to computation efficiency, POD has an efficient communication architecture. A major problem with a conventional tile-based many-core architecture is that we expect the interconnection architecture to consume a high percentage of space and unsustainable energy. For instance, the Raw processor consumes 40 percent of die area in its crossbar and buffers.[8] On the other hand, according to estimates reported for Raw and TRIPS, the interconnection-related circuits and wires can consume approximately 36 and 25 percent of the overall chip power.[9,10] Out of these consumptions, input buffers and crossbar consumed 61 and 68 percent.[11] Without any software hint, it is difficult to apply clock-gating to these systems because the communication patterns are completely nondeterministic.[12]

In contrast, each PE in POD requires only one multiplexer and one demultiplexer for its 2D torus network. A PE requires neither crossbars nor input/output buffers of a conventional packet-switched 2D torus network. The only buffer required is the permutation queue, which is activated only during non-nearest-neighbor communication. In addition to this efficiency, a software-directed communication mechanism along with its deterministic latency makes the 2D torus traffic predictable. Thus, we can use clock-gating to further suppress the energy consumption of wires. Figure 7 shows the active time of inter-PE point-to-point links with respect to the overall execution time for PODs with difference sizes. Although FFT is a well-known communication-intensive application, POD's synchronized computation and communication model makes it possible to disable its point-to-point links for more than 95 percent of the total execution time, thus minimizing the communication links' energy consumption.

Although this work focuses on a dedicated POD layer for one host processor core, we envision that a future 3D-integrated many-core processor will contain multiple general-purpose cores commanding and sharing a common acceleration layer. How to effectively virtualize the common POD among multiple cores requires further investigation in both the architecture and OS designs.

On the other hand, memory bandwidth will only become more limited when the number of cores and the PE array's size are increased with future process technology.

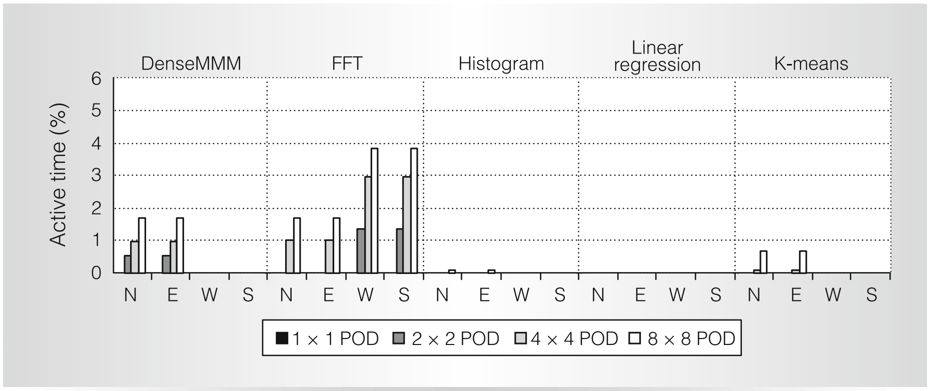

Figure 7. 2D torus network active time. (Only five benchmark applications that use the 2D torus network are shown.)

To approach the best possible energy efficiency, a dynamic monitoring mechanism will be needed for measuring on-chip memory bandwidth consumption and each individual PE's performance. Based on these measurements, the runtime system can more flexibly determine the optimal number of simultaneously active PEs, minimizing the odds of wasted energy. To design a more aggressive system, however, we can use the same 3D die stacking technology to improve bandwidth by integrating additional memory die layers such as SRAM or even DRAM array atop of the POD and the host processor layers.

Last but not least, due to the extremely low latency of interdie vias, the POD layer has the potential, if used intelligently, to improve sequential performance for applications even without any data level parallelism. Several challenges from the architecture down to the technology level must be addressed before these techniques become practical. MICRO

References

1. S. Borkar, ''Thousand Core Chips: A Technology Perspective,'' *Proc. 44th Design Automation Conf.* (DAC 07), ACM Press, 2007, pp. 746-749.
2. W.-M. Hwu et al., ''Implicitly Parallel Programming Models for Thousand-Core Microprocessors,'' *Proc. 44th Design Automation Conf.* (DAC 07), ACM Press, 2007, pp. 754-759.
3. R. Espasa et al., ''Tarantula: A Vector Extension to the Alpha Architecture,'' *Proc. Int'l Symp. Computer Architecture* (ISCA 02), IEEE CS Press, 2002, pp. 281-292.
4. W. Dally and B. Towles, ''Route Packets, Not Wires: On-Chip Interconnection Networks,'' *Proc. 38th Design Automation Conf.* (DAC 01), ACM Press, 2001, pp. 684-689.
5. M.D. Grammatikakis et al., ''Packet Routing in Fixed-Connection Networks: A Survey,'' *J. Parallel and Distributed Computing*, vol. 54, no. 2, Nov. 1998, pp. 77-132.
6. Y. Cao et al., ''New Paradigm of Predictive MOSFET and Interconnect Modeling for Early Circuit Simulation,'' *Proc. IEEE Custom Integrated Circuits Conf.* (CICC), IEEE Press, 2000, pp. 201-204.
7. R. Gonzalez and M. Horowitz, ''Energy Dissipation in General Purpose Microprocessors,'' *IEEE Trans. Solid-State Circuits*, vol. 31, no. 9, Sept. 1996, pp. 1277-1284.
8. M.B. Taylor et al., ''The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs,'' *IEEE Micro*, vol. 22, no. 2, Mar./Apr. 2002, pp. 25-35.
9. J.S. Kim et al., ''Energy Characterization of a Tiled Architecture Processor with On-Chip Networks,'' *Proc. 8th Int'l Symp. Low Power Electronics and Design* (ISLPED 03), ACM Press, 2003, pp. 424-427.
10. L.-S. Peh, ''Chip-Scale Networks: Power and Thermal Impact;'' www.princeton.edu/~peh/talks/stanford_nws.pdf.
11. H. Wang, L.-S. Peh and S. Malik, ''Power-Driven Design of Router Microarchitectures in On-Chip Networks,'' *Proc. Int'l Symp. Microarchitecture* (MICRO 03), IEEE CS Press, 2003, pp. 105-116.

12. S. Borkar, ''Networks for Multi-core Chip: A Controversial View,'' *Proc. 2006 Workshop On- and Off-Chip Interconnection Networks for Multicore Systems* (OCIN 06) 2006; www.ece.ucdavis.edu/~ocin06/talks/borkar.pdf.

Dong Hyuk Woo is a PhD student in the School of Electrical and Computer Engineering at Georgia Institute of Technology. His research interests include energy-efficient many-core architecture and resource-sharing problems of multicore processors. Woo received his MS in electrical and computer engineering from the Georgia Institute of Technology.

Joshua B. Fryman is a senior research scientist with Intel's Microprocessor Technology Lab. His interests include computer architecture, interconnection networks, parallel programming, and the performance bottlenecks of many-core systems. Fryman received his PhD in computer science from the Georgia Institute of Technology.

Allan D. Knies is a principal engineer and associate director of Intel's Berkeley Research Lab. His interests include computer architecture, performance analysis, and parallelism. Knies received his PhD in computer engineering from Purdue University.

Marsha Eng is a marketing engineer at Intel Corporation. Her interests include computer architecture design, performance analysis, and benchmarking. Eng received her MBA from Santa Clara University and MS in computer science from the University of California at San Diego.

Hsien-Hsin S. Lee is an associate professor of the School of Electrical and Computer Engineering at the Georgia Institute of Technology. His research interests include computer architecture, cyber security, and 3D integration. Lee received his PhD in computer science and engineering from the University of Michigan at Ann Arbor. He has received the US Department of Energy and the National Science Foundation Career awards.

Direct questions and comments about this article to Dong Hyuk Woo, School of Electrical and Computer Eng., Georgia Inst. of Technology, 266 Ferst Dr., KACB 2313, Atlanta, GA 30332-0765; dhwoo@ece.gatech.edu.