POD: A Parallel-On-Die Architecture

Dong Hyuk Woo[1] Joshua B. Fryman[2] Allan D. Knies[3] Marsha Eng[2] Hsien-Hsin S. Lee[1]

[1]School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332
{dhwoo, leehs}@ece.gatech.edu

[2]Microprocessor Technology Labs
Intel Corporation
Santa Clara, CA 95052
{joshua.b.fryman, marsha.eng}@intel.com

[3]Intel Research Berkeley
Berkeley, CA 94704
allan.knies@intel.com

1. INTRODUCTION

To integrate thousands of cores on a single die in multi-billion-transistor era, one fundamental limit — power consumption must be addressed. Most of the current multi-core designs leverage off-the-shelf processor economies of scale and focus on symmetric MIMD-style. To not lose the scalability of computation, communication efficiency is to be maintained by employing complex interconnection network or even a network-on-chip. Such design paradigm, however, will become infeasible for a many-core die due to two major power consumers: the massive number of general-purpose processor cores and the increasingly complex interconnection network. Besides, such design is also inefficient in terms of performance per mm^2.

In this abstract, we propose and investigate an alternative architecture in future many-core era. In particular, our thrust is to exploit data level parallelism with high energy/area efficiency while not compromising compatibility of legacy applications. The architecture revisits data-parallel SIMD computers [1, 4] with modern design constraints for a feasible single chip implementation. The overall goal is to provide best-in-class *performance per watt* across the space of many-cores, graphics processors, and media accelerators.

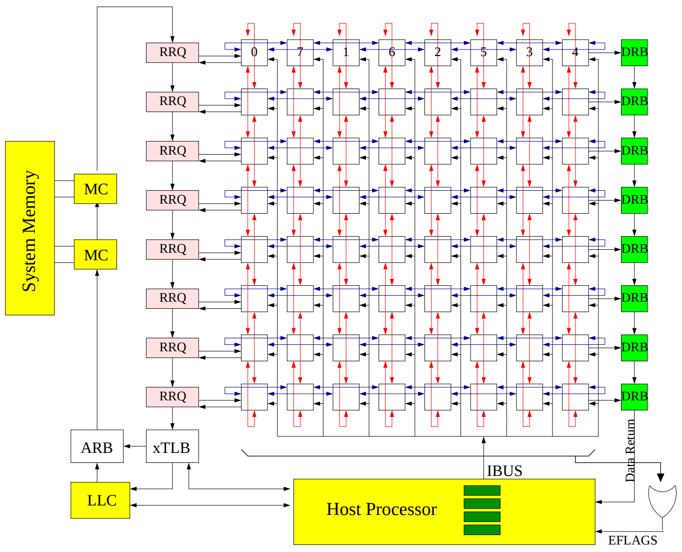

Figure 1: Parallel-On-Die Architecture

2. PARALLEL-ON-DIE ARCHITECTURE

We propose a new massively parallel processor architecture called *Parallel-On-Die* or *POD* illustrated in Figure 1. POD is a fully integrated processing fabric on a single die based on the legacy Intel 64 ISA and provides best-in-class single-stream performance for scalar applications as well as a robust parallel SIMD PE array for scalable parallel application execution. The salient features of POD architecture is summarized as follows.

- **Host Processor Core**

  To maintain scalar performance and backward compatibility for legacy applications, a conventional processor core such as Intel Core 2 Duo is provided in POD as the host processor. In addition to boot an OS and manage system peripherals, the host processor is also responsible for controlling the featured massively parallel computation engine — the *POD SIMD PE array* comprised of an $n \times n$ tiles (8×8 in the Figure). The host processor broadcasts POD instructions (in VLIW format) to be executed on the SIMD array and orchestrates their execution. The generated output can either be delivered to the host processor via explicit instructions or stored directly back to the main memory.

- **SIMD PE Array**

  Each PE tile consists of a high performance SSE-enabled ALU with a local SRAM memory and its own private general-purpose and SSE register files. There is no complex logic such as branch predictors or CISC decoders for achieving low power. The processing core of each PE is a 3-way VLIW that executes a memory, SSE, and integer instruction for each cycle. Each PE also contains four input and four output point-to-point links, all unidirectional to communicate with their neighbors. Under our current programming model, each PE executes the same instruction broadcast by the host processor.

- **POD Interconnection Network**

  As shown in Figure 1, the inter-PE communication is performed via a modified 2D torus network, which is designed to take one uniform cycle traversing across one PE for each communication. The latency is deterministic given its layout. The number inside each PE on the top row indicates the nearest neighbor connection link. These links can be enabled on an as-needed basis as the communication latencies are deterministic, thereby facilitating more power saving opportunities by disabling unused links.

- **Interaction among PEs**

  The 8 point-to-point communication links aforementioned are arranged such that they are glueless drop-in components, with each neighboring PE only requiring direct wiring to complete the layout. This allows for dense packing. Note that neighbor-to-neighbor communication is fully controlled by software and requires neither arbitration nor routing. Thus it removes the need to have buffers for communication, which is known to be highly energy-consuming on packet-switched on-chip interconnect [2]. Each PE can communicate with its neighbor by moving a 128-bit register value or via memory-transfer operation transfer in 64-bit chunks. To enable

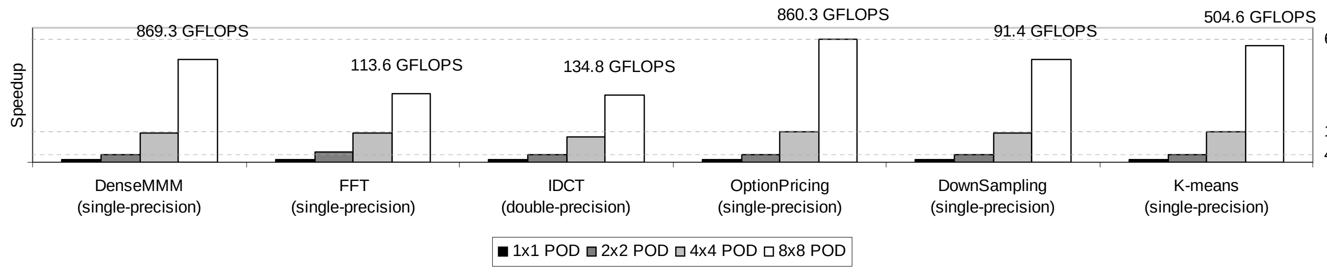

Figure 2: Performance of POD

non-nearest-neighbor, we develop an algorithm based on k-permutation routing [3] in our interconnect design.

- **POD and System Memory Interaction**

  Aside from a local SRAM allocated to each PE, applications are allowed to communicate with the system memory using memory instructions. To manage this interaction, each PE is enhanced with special MBUS to the main memory via an interface called *Row Response Queue* (RRQ). Since system memory operations of all PEs are synchronized, PEs can safely disable their MBUS and related logic when not used for minimizing energy. The RRQ is the queuing point for transactions in both directions, and in turn is connected to a memory ring with the host's last level cache (LLC) and all memory controllers (MCs).

- **Physical Design Evaluation**

  Our implementation is aimed at 3GHz with a 45nm process technology. For an 8×8 POD array each containing 128KB SRAM, we project the peak performance to be 1.5 TFLOPS and 768 GFLOPS for IEEE SP and DP operations. Using datasheet from Intel's 65nm Conroe processor with conservative scaling and their 45nm SRAM cell library, the SIMD pipeline and the local SRAM of each PE are estimated to be $2.1mm^2$ and $0.363mm^2$, respectively. Hence the 8×8 POD array will amount to $205mm^2$. Taking the host core ($25.9mm^2$), the RRQ ($25.6mm^2$), and a 3MB LLC ($20mm^2$) into account, the entire POD processor is approximately $276.5mm^2$.

3. ISA AND PROGRAMMING MODEL

The SIMD execution inside POD is completely managed by the host processor. To enable this, we propose extending the host core with five new instructions and modifying three others. Our new instructions include:

- *SendBits*: broadcast instructions to the POD.
- *GetFlags*: to obtain the return status.
- *DrainFlags*: assure that the initial setup of a known state in the flag tree is complete.
- *SendRegister*: broadcast a host register to every PE.
- *GetResult*, to obtain a return buffer value from the POD without using system memory as a go-between.

The three modified host instructions are the various *fence* operations (Load, Store, and combined) extended to monitor the return status of the POD's memory interface.

The POD instruction streams are embedded within the regular Intel 64 binaries. A *SendBits* instruction is used for the host to forward a 12-byte VLIW instruction to the POD array. POD instructions are transferred by the host in a pipelined fashion. Hierarchical buffers were implemented in each row of the POD array to make the same instructions arrive at the same time for all rows.

To support multi-level conditional execution (e.g. nested if-then-else) in the PE, two types of masking instructions, pushmask and popmask, are provided. They keep track of the nested conditional state. Upon entering each conditional region, pushmask shifts down all the bits in the mask register for each PE and sets its MSB based on the test condition. popmask pops the MSB bit out of the mask register as a result of exiting a conditional region.

4. PERFORMANCE EVALUATION

We evaluated the performance of POD architecture with several data-parallel applications using a simulator. Figure 2 shows achieved GFLOPS and relative performance improvement normalized to the performance of 1×1 POD as the number of PEs increases. The off-chip DRAM bandwidth is assumed to be 4×32 GBps (four on-chip memory controllers where each can provide 32 GBps bandwidth) with the DRAM latency as 50 ns. To factor out performance improvement due to larger on-chip memory as the number of PEs increases, we assume that aggregate size of the on-chip memory remains the same regardless of the number of PEs. For example, in our simulations, a PE of 1×1 POD has 8MB of local SRAM, while each PE of 8×8 POD has 128KB SRAM only.

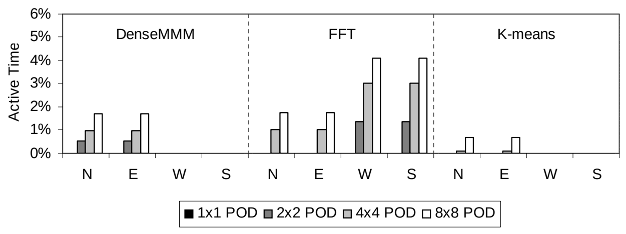

Figure 3: Point-to-point Links Active Time

Figure 3 shows the active time of inter-PE point-to-point links with respect to the overall execution time for difference sized PODs. We show only those applications that require inter-PE communication. As shown, although FFT is communication-intensive, synchronized computation and communication model of POD makes it possible to disable its point-to-point links for more than 95% of the time, thus minimizing the interconnect energy, which will be impossible to do in MIMD-based many-core architecture due to the unpredictable nature of their interconnection.

For more information, please visit our project website at http://arch.ece.gatech.edu/pod.html.

5. REFERENCES

[1] T. Blank. The MasPar MP-1 Architecture. In *Proceedings of COMPCON*, Spring 1990.

[2] S. Borkar. Networks for Multi-core Chip–A Controversial View. In *2006 Workshop on On- and Off-Chip Interconnection Networks for Multicore Systems*, 2006.

[3] M. D. Grammatikakis, D. F. Hsu, M. Kraetzl, and J. F. Sibeyn. Packet routing in fixed-connection networks: A survey. *Journal of Parallel and Distributed Computing*, 54(2):77–132, 1998.

[4] L. W. Tucker and G. G. Robertson. Architecture and Applications of the Connection Machine. In *IEEE Computer*, August 1988.