# Accelerating OpenMP programs by embedding a 1K-core true shared memory architecture on the FPGA.

Yosi Ben Asher, CS Department, Haifa University, Israel, yosi@cs.haifa.ac.il

Manycore shared memory architectures that use many homogeneous small power-efficient cores hold a significant premise to speed up OpenMP parallel programs. Currently shared memory is realized by maintaining cache-consistency across the cores, caching all the connected cores to one main memory module. This approach, though used today, is not likely to be scalable enough to support a 1K-core architecture. Therefore we consider a known theoretical scheme for shared memory machines wherein: the shared address space is divided between a set of memory modules; and a Butterfly communication network (BN) allows each core to access every such module in parallel. Congestion in the BN and on the access to the memory modules is resolved by rehashing the memory address-space. We used simple power efficient techniques to simplified some of the practical aspects involved with realizing this theoretical construction, e.g.: overcoming collisions at the BN when multiple memory references are made in the same cycle and implementing the BN as one combinatorial circuit rather than as a multi-stage network. The proposed architecture has been synthesized it to $2, 4, 8, 16, \ldots, 1024 - cores$ for an FPGA and was evaluated for several parallel programs. The synthesis results and the execution measurements show that, for the FPGAs, all problematic aspects of this construction can be resolved. For example, unlike ASICs, the growing complexity of the communication network is absorbed by the FPGA's routing grid and by its routing mechanism. This makes this type of architectures particularly suitable for FPGAs. In this research we propose to develop this 1K core architecture for FPGA (Xilinx and Intel's Xeon+Arria10 machine) including: 1) Improved architecture that will be based on CPU0 cores, 2) An LLVM based OpenMP compiler, and 3) a runtime OpenMP library supporting OpenMP threads and communication with the main program running on the CPU. We thus intend to show that the FPGA can be used to significantly speedup the execution of OpenMP programs executed on a host machine.

## 1 Proposed architecture

The proposed multicore contains the following components (as depicted in figure 1): item A set of $n = 2^k \quad k = 0, \ldots, 10$ memory modules $M_1, \ldots, M_n$ form the memory shared address space of $n$ cores. These Each core has call/return stack and an instruction ROM that are managed through local memory modules. Memory references are generated as packets where each packet contains the following fields:

$$[R/W - bit \mid memory - module \mid internal - address \mid priorety]$$

A multistage butterfly topology network (BN) connecting the $n$ memory ports of the cores to the shared memory modules. The BN works as a combinatorial circuit allowing packets to be routed to their destination in one clock cycle (unless collided with another packet). Figure 2 illustrates collision of two packets attempting to use the same exit of a switch where one sent from core 0010 to module 1010 and the other from 1011 to 1000. When a packet is dropped at a given switch the core that issued this memory reference is blocked and the memory request is repeated in the next clock cycle until the packet reaches its destination. Note that the failing signal is not propagated directly to the core but through the backward BN's switches. Figure 1 illustrates the proposed architecture for $n = 8$ cores ($C_0 \ldots C_7$), showing the BN (forward and backward), partition of the shared address space to memory modules, the hashing units, the ring-buffer with two real ports and the interrupt-buss. When a packet $pa$ is dropped in a switch $S_{i,j}$ it is stored in $S_{i,j}$ in

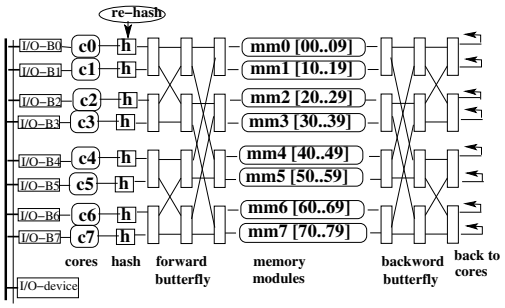


Figure 1: proposed architecture

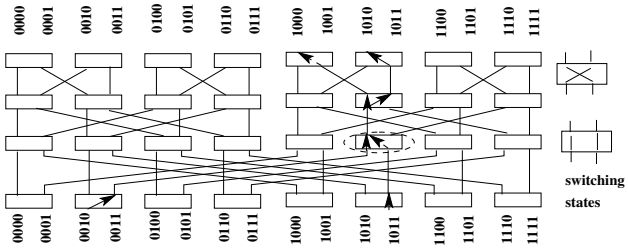


Figure 2: BN and a collision

an intermediate-register and will continue in the next clock cycle, not from the original core that issued it, but from $S_{i,j}$. This complicates the logic at each switch but can: a) Make free the switches in the path of $pa$ to $S_{i,j}$ so that they can be used by other packets (e.g., $pe$s in the figure). or b) Prevent a possible "left blocking" of $pa$ at the next clock cycle by a packet $pc$ with a higher priority. Collisions (as depicted in figure 2) can now occur not only between two packets entering the BN's but also with the packet that resides in the intermediate register. In this case (three packets collide) one of the packets is dropped back to the core that issued it, one stays in the intermediate register and one continues to the next level. The decision which packet is dropped and which continues is made based on a time+source priority mechanism of the packets arriving to the switch. The theoretical framework requires that one hash-function should be selected in random from a family of $n$ universal hash functions before a program is executed. This is not practical, we have tested a smaller family of such functions that can be realized without increasing the clock latency or add extra clock cycles to load/store operations. It trivially follows that there is no starvation and the priority mechanism that was implemented creates a fair execution since packets that collide and are dropped (back to the core or to an intermediate register) acquire a higher priority.

# 2 Preliminary experimental results

We have designed and implemented an preliminary version of the proposed architecture to obtain proof of concept for the general idea. Programming was achieved by using a simple C compiler for the Picoblace core and extending the resulting assembly to an SPMD programing mode to create executable for each core. This was done ad-hoc using editing-scripts of the generic assembly code. The architecture was implemented in Verilog such that by modifying some constants it can be compiled to different value of the number of cores. We used 32-bits modified PACOBLAZE cores and tested different parameters of this architecture verifying its ability to achieve high speedups. The results of the FPGA synthesis (using Xlinx Vivado) of this architecture are depicted in the following table showing that the clock latency does not increase significantly when more cores are used. The fact that the resources (LUTs+Registers) at most doubles every time we double the number of cores prove that the connections between the BN-switches is absorbed by the FPGA. It thus follows that the BN can be embedded by the routing infrastructure of the FPGA without increasing the memory latency. Though there is a small decrease in the clock frequency when more cores are used it is negligible compare to the double increase in the number of cores. Note that these results, for 1024 core, are close to what is possible on the Arria-10 GT 1150 and we believe we can further reduce it in the proposed research. In addition we obtained high speedups for a set of algorithms such as FFT,FIR,N-BODY and SORTING. For example, for the Viterbi algorithm we obtained a speedup of 887 compare to a sequential run using a 1024-core configuration.

| Cores | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|---|---|---|---|
| freq. Mhz | 101 | 99 | 98 | 96 | 95 | 95 | 95 | 87 | 86 | 83 |
| LUTs | 1854 | 4350 | 8621 | 16785 | 37274 | 81216 | 171731 | 363975 | 756602 | 1813814 |
| Registers | 237 | 482 | 979 | 1988 | 4037 | 8198 | 16647 | 33800 | 68617 | 139274 |
| DSP | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | 3568 |