

Algorithms in Hardware – Parallel Logic Structures

By Phil Bording

Computer Science Chairperson
Alabama A&M University

The challenge of the future in computer hardware design is one of how to use the available logic resources within a single chip. Until recently the engineering effort has been to concentrate all of the logic functions from separate chips and integrate them into the microprocessor. The last hold out in this cost driven engineering process is the graphics “card”. The high end graphics processor has had a power and cooling requirement large enough to be kept as a separate chip within the design. However as transition in manufacturing to smaller line widths – now much less than 100 nanometers – is taking place the last holdout will fall. Both Intel and NVIDIA are driving to a single chip solution. NVIDIA has announced that they will have a x86 style architecture and Intel has been producing micro’s with ever more powerful graphics capability. With announced products of 500 gigaflops or more, NVIDIA has set the mark for peak performance on a single plug-in board to a PC. The question remains how effective will these programmable products be?

The single benchmark, LINPACK (with matrix-matrix operations it is called LAPACK) has been used to validate the effectiveness of high performance computers. Linear equation solvers have $O(n^2)$ input/output and $O(n^3)$ operation counts with very high arithmetic demands for every memory operation. Thus LINPACK does not represent the class of algorithms that are balanced, in the sense that memory operation and arithmetic operations are nearly equal. Much of scientific computation and the supporting numerical approximations are reasonably balanced in memory / arithmetic operations. Hence, these algorithms can make use of the current cache memory support in microprocessors but typically do not perform at peak speeds like LINPACK. Real world examples of graphics card processors (GPU’s) show factors of two to four improvements a far cry from the two orders of magnitude within the *peak*. So what is the future?

Currently a single chip microprocessor has a single memory bus with an exceptionally wide data bus to provide a single cache line load every clock. The use of high speed double and triple data rate clock schemes has pushed the memory bandwidth to numbers in excess of a gigahertz. The bottleneck is the dynamic ram (DRAM) cell latency. The DRAM capacitor has to be charged and discharged to be read and the associated device physics time constant does not scale well as the ever-changing technology makes the parts smaller. The future is more memory paths to the microprocessor – each being a cache line in width. These multiple memory paths will have address and data paths with line counts in the hundreds, with the resulting chip will have every increasing pin/pad counts. So what could we do differently?

Well much of what we do in computing is repetitive, and we could examine these high usage algorithms for parallel constructs and build devices which are dedicated. We could use the 100 million – billion transistor budget in the microprocessor to do the algorithms needed for solving a specific problem or class of problems. In fact, we could leave out the microprocessor part and just build an application

specific machine. Much of the current difficulty in high performance computing (HPC) is in the operating system and compiler software. Building a dedicated hardware machine does take time, but it won't have to wait for the compilers and operating systems programming effort to be useful. The landscape of HPC systems is littered with working hardware for which the software was never truly finished. So can we really build a machine that works as a system without using a commercial microprocessor? Probably not, but we can build problem specific memory and arithmetic subsystems that have a serious set of data and address paths so that balanced algorithms work efficiently. These parallel logic and memory structures are the key to these application specific machines. I have examples of machines for solving partial differential equations using both finite differences and finite elements, for level set algorithms and for the ever present Fourier transform problem. To delve into these designs takes some time so we can understand the problem and then the solution.