Evaluation of Large Matrix Operations on a Reconfigurable Computing Platform for High Performance Scientific Computations

Chen Chang, Gerald Wang, David Culler
University of California, Berkeley
{chenzh, geraldw}@eecs.berkeley.edu, culler@cs.berkeley.edu

Abstract

This paper evaluates the performance of floating point matrix operation on a large-scale Field Programmable Gate Array (FPGA) based Reconfigurable Computing Platform (RCP) using Linpack benchmark. A baseline performance was established on an existing multi-FPGA array platform, then the effects of a set of system architecture parameters, such as interconnect topology, latency, bandwidth, memory capacity and bandwidth, are evaluated to determine the overall system performance sensitivity. Finally, based on the sensitivity results, a set of design parameters was chosen to maximize performance while constrained by the current technology for a 6561-FPGA system, and the projected performance was determined to be 46.9 TFLOPS, at the Linpack matrix dimension of 4,408,992.

1 Introduction

In the application domain of high performance scientific computing, large dense matrix operations are at the heart of many computation kernels. Thus, efficient massively parallel floating-point number operation is one of the key factors in reducing the overall application execution time. To increase the parallelism in the system, one might simply increase the total number of available processing elements in the system. However, due to communication and synchronization overhead, the incremental performance gain from the increase of processors quickly diminishes. By using ever increasingly complex interconnect topology and connection technology, one can marginally increase the performance for a given size of a system. Nevertheless, to build a system that can deliver Tera-FLOP range performance, the cost of the exotic interconnect, the complexity of testing, programming, and maintenance have driven recent generations of supercomputers well beyond hundreds of millions of dollars, even though most of the systems utilize commonly available cheap off-the-shelf processors and memory modules.

Although dense matrix kernels typically have a very regular and predictable memory access pattern, which leads to high cache hit rate, sharing a large amount of memory space covered by the kernel requires a much larger memory bandwidth than typical applications. Since most off-the-shelf processors are designed to perform the best for typical applications, they usually do not have enough continuous memory bandwidth to sustain the on-chip high clock rate floating point units, thus reducing the actual sustained throughput at a fraction of the peak performance. Furthermore, since the processors in the system operate asynchronously with each other, costly explicit synchronization and barrier operations can reduce the overall performance by a factor of two to three.

Ideally, in a system with as few individual processors as possible, each processor has an as high number of parallel processing units as possible and matching memory bandwidth to sustain the continues operation, and all processors are synchronous with each other with compile time predictable cycle-accurate execution time, will be able to perform the best for large dense matrix operations. Although, such a system does not exist in practice, many of the characteristics are very similar to FPGA-based reconfigurable computing platforms. FPGA chips typically run at a couple of hundreds of MHz, and offer a large amount of parallel I/O pins (currently up to 1200 for Xilinx Virtex II Pro series [1]). Not only the clock rate matches nicely with the current DRAM technology, but also the large I/O pins offer the solution to the high memory bandwidth requirement by supporting up to 16 independent 8-byte-wide DDR memory channels. In addition, each FPGA can functionally emulate up to two million ASIC (Application Specific Integrated Circuit) equivalent designs, which makes it easy to construct a large number of on-chip parallel processing units, running currently, and each dedicated to a portion of the overall algorithm. Thus, by matching the implementation hardware architecture to the application, all levels of parallelism existing in the original algorithm can be fully exploited. When a fully synchronous programming model is utilized, the exact number of execution cycle for each processing unit can be determined at the compilation time, therefore avoiding the costly synchronization operations.

The rest of the paper describes an experiment of implementing the Linpack benchmark on a RCP to maximize the overall performance. Section 2 gives a brief overview of existing solutions for large matrix operations,
and explains the RCP advantages and limitations. Section 3 details the experiment setup and the particular benchmark algorithm used. Section 4 first analyzes the performance on an existing multi-FPGA platform, and then explores the effect of a set of different system design parameters on the overall performance; finally, the scalability of this approach is illustrated through the design of a 6561-FPGA system.

2 Background Materials

2.1 Existing Solutions

General-purpose parallel high performance computing clusters are very attractive for large dense linear systems where computation is intensive. Domain dissection introduces desired high-level parallelism into matrix algorithms. A coefficient matrix is dissected one or more times to produce a set of sub-matrices, and a sub-matrix is then fed to a parallel processing unit. We focus on Gauss Elimination (GE) since it is a widely used direct method for solving such linear systems. The GE algorithm [2] consists of two phases. The first phase is elimination wherein the matrix is converted to an upper triangular, followed by back substitution to compute the values. For a dense matrix of order \( N \), the computational complexity is \( 2N^3/3 + O(N^2) \).

To improve numerical stability the use of partial pivoting is mandatory. In each step, proper multiples of the pivotal row are subtracted from the rows below in order to zero out matrix coefficients. Pivoting requires interchanging rows of the matrix and therefore, in general, extra communication between processors. Also the search for the pivotal element in consecutive stages of the factorization process requires inter processor communication. As a consequence, the use of partial pivoting results in increased data traffic between processors. Experiments in [3] show that threshold pivoting is a valuable mean to improve the performance of parallel GE by cutting down on communication time while retaining the stability of the algorithm.

Introduction of pipelining into the back substitution phase of GE is investigated in [4]. During this phase, each value depends on all the previously calculated values. This particular data dependency pattern expresses a particular kind of communication and a form of pipelining. By splitting the communication and calculation into several parts to overlap them, the parallel behavior is improved.

The total cost of a parallel version of GE algorithm is composed of local computation cost, communication cost, and synchronization cost. Algorithm design aims to minimize them, but it remains an open question whether the optimal communication and synchronization costs can be achieved simultaneously [5]. Parallel architectures have memory limitations and communication bottlenecks, and they suffer long synchronization delays. Cache design is still a big challenge because of cache coherence and cache consistency problems [6]. Due to the nature of general purpose, the power consumption is high.

On the other hand, specialized high performance computer systems are used to deal with computations in linear systems, which are burdensome to general-purpose computers. Systolic architectures are based on the concept of systolic arrays. The representative proposed methods include Leiserson method [7], Double Pipe Method [8], Fitted Diagonal Method [9], Folding and Rotating Method [10], and Unidirectional Method [8]. In [11] the authors conduct some interesting theoretical analysis on their proposed new systolic array and claim that their design has a better time complexity. All systolic architectures use pipelining to exploit parallelism and to achieve high performance. However, partitioning and scheduling of data arrays for systolic computation are generally difficult and performed ad hoc. In addition, the high I/O bandwidth requirement can drastically reduce the overall performance speedup when used in larger systems [12].

2.2 Reconfigurable Computing Platform

Most RCP systems are constructed using FPGA chips, which are locally interconnected to their neighbors [13]. The basic idea is to use the off-chip interconnects to form an aggregated virtual FPGA that has enough capacity to implement the target application. Since currently available commercial FPGAs are designed with a mesh-type on-chip network to retain as much homogeneity as possible. Thus mapping and routing software are simplified. RCP systems typically use 2-D or 3-D mesh or torus as their main high-bandwidth inter-FPGA connection topology.

Unlike the traditional processor-based systems, in RCP systems, application algorithms are mapped to the processing fabric both temporally and spatially. Not only different FPGA’s can be dedicated to operate on different processes independently or constructively, but also within the same FPGA, different parts of the chip can operate on various operations from the granularity of individual logic gates to the entire chip. Therefore, all levels of intrinsic parallelism existing in the application can be fully exploited to maximize performance. When used in conjunction with a global synchronous programming model, different FPGA’s can operate synchronously with one another, and the exact execution cycle can be determined at the compilation time, rather than leaving to run-time, thus foregoing much synchronization overhead.

Under the pressure of ever increasing complexity in sub-micro fabrication processes, in recent years, with its regular structure, the FPGA capacity has increased faster than the
traditional Moore’s Law for microprocessors, and become the fabrication technology leader in the semiconductor industry. Instead of pushing the on-chip clock rate to GHz range, FPGA’s remain at a few hundreds of MHz, mainly due to the dense programmable interconnect structure on-chip. However, FPGA’s provide spatial parallelism with an order of 100s to 1000s rather than the typical 10s found with traditional scalar processors. Therefore, the overall parallelism on FPGA’s is between 10 to 100 times more. By trading spatial parallelism for temporal parallelism, the power consumption of FPGA’s is also less than 10% of that of typical general-purpose processors found in PC’s or servers.

Due to the lack of a widely adopted general purpose programming model and a necessary component virtualization, such as virtual memory, most RCP systems are currently used as custom computing machines each designed to satisfy the particular needs of each application or to narrow application domain. In particular, in DSP or communication application domain, where the application algorithms exhibit a stream-based data flow through a large number of locally connected processing elements with occasional feedbacks and little control logic, RCP solutions have been proven to accelerate the emulation of the target application by a factor of 1,000 to 1,000,000 times faster than the general-purpose processor solution [14]. However, almost all of these applications utilize fix-point number representations of eight to 16 bits wide, thus the operators take much less area and requires much fewer pipeline stages than that of an IEEE-754 compliant single or double precision floating point operator. Although a few of previous research have shown that implementing floating-point operators on FPGA is feasible [15-20], most of them fail to show that doing so actually can result in performance improvement over general purpose processor solutions, mainly because the previous generations of FPGA technology cannot fit more than a few floating point operator on-chip. However, as will be shown later in the paper, the current FPGA capacity has grown large enough to fit up to 100 single precision FP operators on-chip, thus producing more throughput than general purpose processor solutions.

3 Experiment Setup

3.1 Linpack Benchmark

Designed in the 1970s and early 1980s, Linpack [21] is a collection of FORTRAN subroutines that analyze and solve linear equations \(Ax = B\) and linear least-squares problems. Linpack has been widely used to evaluate peak performance of high performance computing solutions. To evaluate large matrix GE with partial pivoting and back substitution, the coefficient matrix \(A\) is randomly generated and vector \(b\) is constructed in such a way that the solutions for the variable vector \(x\) are all ones.

In this project, a parallel version of GE algorithm with backward substitution for a reconfigurable computing platform is designed. In this algorithm, a cyclic mapping approach is chosen to assign equations to different parallel processing units (FPGA’s). During the elimination step, pivotal elements’ values and the corresponding pivotal row are broadcast, while during the backward substitution step, only the computed variables’ values are broadcast. To overlap local computation and communication as much as possible and get more parallelism, the right hand side vector of the linear equations is updated right after an unknown’s value is obtained. The details of the design parameters will be explained in the next section.

3.2 Design Parameters

There are two types of parameters related to our experiments: linear system problem related and architecture platform related. The matrix dimension \(N\) is the only system related parameter, which captures the size of the system we are solving.

Our reconfigurable computing platform is FPGA-based. The parallelism of the system, denoted by \(K\), represents the number of parallel units. The parallel units per FPGA are represented by parallel factor \(F\). Therefore the value of \((K/F)\) is the total number of FPGA’s in the system. On top of the connected FPGA’s, FPGA clusters may be formed by grouping \(C\) FPGA’s together. Three types of inter-FPGA connection topologies, 2-D mesh, 2-D torus, and C-ary tree as discussed in [22], are studied in this project. An average hop model is used to model link delays for them:

- 2-D mesh: \[\frac{4}{3}\sqrt{(K/F)}\]
- 2-D torus: \[\frac{1}{2}\sqrt{(K/F)}\]
- C-ary tree: \[\log_C^{(K/F)}\]

Parameter \(\text{link\_delay}\) captures the cycle delay between FPGA’s. Clock rate \(R\) is the target system speed including all on-board and inter-chassis delays. Given the current interconnect technology, the value is about 100 MHz.

Two different broadcast methods are used for communication: fixed-delay based and C-ary tree-based. The broadcast cycle delay for C-ary tree-based is:

\[2\log_C^{(K/F)}, \text{link\_delay}\]

There are four types of floating-point operation: floating-point add, multiply, comparator, and divide. In order to sustain a clock rate of 100MHz, except divide, the other three operations are pipelined. The parameter
As will be explained later, divisions are done rarely in comparison to multiply and subtraction in the Linpack algorithm, so an iterative divider is used to save area, while still is able to maintain throughput. Parameter $\text{divid\_cycle}$ is the fixed number of division iteration cycles.

3.3 Evaluation Method

Among the three broad techniques [23] for performance evaluations, system-level cycle-accurate simulation is used for rapid exploration of system design parameters, while maintaining hardware implementability. Subcomponents, such as floating point operators and control logic, are modeled directly based on VHDL hardware synthesis results using Synplify Pro [24] as the synthesizer. Detailed micro-architecture assumptions and hardware results are presented in the next section.

4 Analysis

4.1 Micro Architecture

As explained in section 3.1, the GE algorithm consists of the following five steps: 1) pivot row selection, 2) pivot row swapping, and broadcast of the pivot point value, 3) pivot column division, 4) sub-matrix elimination, and 5) back-solve.

In pivot row selection, a binary floating-point number comparison tree is needed to choose which row of the sub-matrix has the largest absolute value in the pivot column. As shown in table 1, each of the 32-bit single-precision floating-point comparator occupies 100 LUTs (Look-Up Tables) and requires two pipelining stages to run above 100MHz. The total comparison tree delay consists of three parts. The first one is the pipelined comparator delay. In each $K$ vector processing, the sub-tree height is $\log_2 K$, and in the $i$th elimination iteration, there are $\lceil (N-i)/K \rceil$ batch comparisons. Since each batch is skewed by exact one cycle, the max value and corresponding row index only have to be kept at the root of the K-leaf comparison tree. Further more, the K leaves span $\lceil K/F \rceil$ FPGA chips. When using a C-ary aggregation tree of height $\log_C(K/F)$ to merge the results, the total inter-FPGA link delay is $\log_C(K/F) \cdot \text{link\_delay}$. After the pivot row has been chosen, the decision needs to be broadcast to the leaf nodes to setup the row swapping operation. If fixed-delay broadcast method is used, the broadcast latency is just $\text{bcast\_delay}$. If C-ary broadcast tree network is used, then the latency is $\log_C(K/F) \cdot \text{link\_delay}$. Then the total pivot row selection operation latency is just the sum of the above four latency components.

<table>
<thead>
<tr>
<th>Operator</th>
<th>LUTs</th>
<th>Async Delay (ns)</th>
<th>Pipeline Stages</th>
<th>Latency (cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiply</td>
<td>1357</td>
<td>51.7</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Subtract</td>
<td>1155</td>
<td>64.5</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Comparator</td>
<td>100</td>
<td>14.6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Divide</td>
<td>237</td>
<td>9.7</td>
<td>0</td>
<td>27</td>
</tr>
</tbody>
</table>

During pivot row swapping, for the $i$th elimination iteration, the length of the row is $N-i+2$, thus the same number of clock cycles to be put on the inter-FPGA network. Depending on the network topology and the inter-FPGA link delay, $\text{avg\_hops} \times \text{link\_delay}$ additional clock cycles are needed, where $\text{avg\_hops}$ is the average number of hops between any two points on the network, which depends on the network topology. Assuming a full duplex network link, the swap can happen concurrently directly between the two memory modules, which may be attached to the same or different FPGA’s. After the swap, the pivot element value needs to be broadcast to the rest of the FPGA chips, thus it requires one more broadcast latency as calculated above.

Since the pivot column division occurs only once per sub-matrix elimination iterations, there is no need to dedicate a fully pipelined divider. The division of the pivot column of the next iteration can be overlap with the current iteration, as soon as the second column elimination step completes in the current iteration. Therefore an area efficient divider with a large number of iteration cycles can be used instead. In this case, a 237-LUTs 27-cycle iterative divider is used.

The sub-matrix elimination is the most computationally intense part of the whole algorithm, thus, the parallel floating point multiplier and subtractor are fully pipelined to achieve a clock rate of 100MHz or more. Since on Xilinx FPGA’s, each LUT is followed by a register, it costs little extra LUT to moderately pipeline the original asynchronous design. Similarly in the back-solve step, the same multiplier and subtractor operators are used to solve for the x vector.

Overall, although the above five steps seem to be specified by the algorithm to operate sequentially in this order, the order can be changed and operations can be overlapped to further improve the performance. The back-solve definitely needs to occur after all the elimination steps are done, but during the elimination iterations, steps
1) ~ 3) can be overlapped with step 4). During the same time of the elimination of the first non-pivot column, which gives the value of the pivot column of the next iteration, the comparison operation can start simultaneously with the elimination operation, and then followed by the broadcast of the pivot row choice and the row swapping index. If row swapping is overlapped with the elimination step, the memory bandwidth needs to be doubled to sustain continuous operations. Thus, the next iteration pivot row division is allowed to precede the actual row swapping. As long as the order of row swapping and the elimination is maintained, moving pivot row division before actual row swapping does not change the result. In this way, only when the sub matrix becomes sufficiently small, the swapping and elimination step will cost fewer cycles than the rest of the concurrent steps, which causes the stall of the elimination pipeline.

4.2 Baseline performance

To establish a reference point for performance comparison, the particular Linpack algorithm for RCP described in previous sections was first verified on an existing multi-FPGA array, the Berkeley Emulation Engine (BEE) [25, 26]. BEE has two levels of mesh network. On the first layer 8-way mesh, there are 16 Xilinx XCV2000E FPGA chips, arranged in a 4x4 array. As shown in Figure 1, on the second layer, four more FPGA’s are used as crossbar (XBAR) for global control and communication. Therefore, the 16 first layer FPGA’s are used as processing elements, while the four XBARs are used as control aggregation and broadcast medium.

With the emulation capacity of 38,400 LUTs, each of the processing FPGA’s contains 12 parallel processing units; each consists of one parallel floating point multiplier, one subtractor, and one iterative divider as described in the previous section. The comparison tree and the rest of control logic are shared among the 12 processing units on the same chip. Although each processing FPGA owns one Mbytes off-chip SRAM available, due to its relatively narrow 32-bit data interface, for simplicity, only on-chip memory blocks are used in the experiment. Given the on-chip memory capacity of 64 Kbytes per FPGA, a matrix size of 480 was used to span all 16 FPGA’s.

Despite all the on-chip floating operators are pipelined to run above 100 MHz, in the BEE system, the overall clock rate is limited by the large PCB board, where the inter-FPGA link speed is at maximum 60 MHz. Any FPGA on the first mesh layer can communicate with its neighbor in one clock cycle and with any other FPGA’s on the board in 3 cycles through the XBAR network. Therefore, a fixed broadcast method through XBARs brings a 3-cycle broadcast latency.

![Figure 1: Second layer XBAR mesh](image)

![Figure 2: Baseline performance for various matrix sizes](image)
sub-matrix elimination step. Therefore only the row-swap step is visible in the overall execution time.

The other distinct observation on Figure 2(a) is the saw tooth-shaped curve. Since the algorithm used here runs all \( K \) parallel processing elements in the system simultaneously, when matrix dimension \( N \) is integer multiples of \( K \), then there is fewer number of cycles wasted in each of the \( K \) vector processing cycles. Further more, due to the row decomposition scheme used for making sure that the processing units only require local memory contents, even when the sub-matrix dimension is an integer multiple of \( K \), the actual execution will still require more cycles, for the data at the end column are most likely always to spill into the next section of the \( K \) by \( N+1 \) section of the memory, reducing the computation efficiency on the last rows of the sub-matrix. This effect becomes more clear in Figure 3, where the execution cycles for different matrix sizes are normalize to 100%, and the cycles due to elimination, interconnect delay, back-solve, and pipeline stall are displayed individually. In this case, the value of \( K \) is 192, and clearly it matches the periodicity of the saw tooth shape in the percentage plot. Furthermore, every time the value of \( N \) increments by \( K \), the percentage of the interconnect cycles decrease, and the elimination cycles increases. This effect is most pronounced when \( N \) is small, and saturates when \( N \) is above 10 times of the \( K \) value. In comparison to the elimination and interconnect cycle percentage, the portion of the execution for back-solve and pipeline stall becomes negligible as soon as the \( N \) value becomes a few times larger than \( K \). This is mainly because the pipeline stalls are most pronounced in the back-solve stage, where the solution of \( x \) depends on the value of \( b \) in the previous iteration. However, the back-solve only has an order of \( N^2 \) operations, rather than the elimination, which has an order of \( N^3 \) operations. Therefore, for large matrix GE operations, the long pipeline needed for the floating-point operators has little effect on the overall performance.

Although the baseline performance on BEE gives a realistic reference point for comparison, the BEE was originally designed for communication and DSP system emulation, rather than high performance scientific computation. The lack of large high-bandwidth off-chip DRAM and low board-level interconnect speed due to the large board size (53cmX58cm), has hindered performance. In addition, the FPGA chip used on the BEE is two generations older than the current state-of-the-art FPGA. Therefore, in the rest of the analysis, the board-level interconnect is assumed to achieve 100 MHz, and off-chip large-capacity and high bandwidth DRAM modules are assumed attached to each FPGA chip.

4.3 Broadcast and Interconnect Effects

Figure 4(a) illustrates the effect of the broadcast delay on the total execution cycles. The parameters used in the experiment are: \( N = 1000, F = 10, C = 4, K = 100 \) and the latency for C-ary tree based broadcast is fixed at two cycles. The experimental result shows that up to a broadcast delay of 12 cycles, both fixed delay and 4-ary tree-based broadcast approaches outperform 2-ary tree-based broadcast. But for the range of the broadcast delay cycles from one to eight cycles, fixed delay broadcast is better than 4-ary tree-based broadcast. When the broadcast delay is longer than eight cycles, 4-ary tree-based broadcast gives a better performance. In Figure 4(b), the corresponding interconnect cycle percentage is plotted.

Link delay effects are shown in Figure 5. In the experiment, a tree-based broadcast method is used and the rest of parameters remain the same as those in Figure 4. The experimental results show that the number of total execution cycles is insensitive to the interconnect
topologies for link delay from one to 12 cycles. Clearly latency effect is dominated by broadcast. In Figure 5(b) shows the corresponding interconnect cycle percentage change versus link delays.

\begin{figure}[h]
\centering
\includegraphics[width=0.7\linewidth]{figure5.png}
\caption{Link latency effects}
\end{figure}

**4.4 Memory and Bandwidth Requirements**

The memory and bandwidth requirements can be theoretically analyzed. Since all coefficient matrix entries need to be stored in off-chip memory, there must be $4N^2/(K/F)$ bytes off-chip memory per FPGA. There is a higher on-chip memory requirement during back substitution than elimination since both $x$ and $b$ entries need to be kept. The on-chip memory requirement can be calculated by $8N/(K/F)$. Clearly the off-chip memory requirement changes quadratically with the size of the linear system, while the on-chip memory requirement changes linearly with $N$. Figure 6 plots the memory capacity requirement under different problem sizes and different number of FPGA’s in the system.

Communication is needed to broadcast pivotal values and to swap rows for partial pivoting during elimination phase, while the $x$ value needs to be broadcast as soon as it is calculated during the back substitution phase. Theoretical analysis shows that the off-chip DRAM bandwidth requirement is $8FR$ and the on-chip SRAM bandwidth requirement is $4FR$, as shown in Figures 7. Also the current technologies of DRAM and SRAM are plotted. Comparison shows that if $F < 32$ the DDR400 (8 DIMMs) can meet the requirement and if $32 < F < 47$ DDR400 (12 DIMMs) can provide the required bandwidth. For the on-chip SRAM bandwidth requirement, it shows that ZBT with four banks can only meet the requirement for $F < 3$, while XC2VP 100 can provide the required bandwidth for up to $F = 100$.

\begin{figure}[h]
\centering
\includegraphics[width=0.7\linewidth]{figure6.png}
\caption{Memory capacity requirement}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.7\linewidth]{figure7.png}
\caption{Memory bandwidth requirements and current memory technology comparison}
\end{figure}

**4.5 FPGA Capacity and Clustering Effects**

Parameters $F$ and $C$ capture FPGA’s capacity and cluster size respectively. We simulate the system under cluster sizes from 2 to 16 with an interval of 2 and an FPGA capacity from 8 to 48 with an interval of 4. Figures 8 and 9 show the total number of execution cycles and the corresponding contour plot for $K = 100$ and 200, respectively. The common characteristics for them are that
the smaller $F$ and $C$ are, the denser of the contour plot is. For different $K$ values, there are different proper combinations of $F$ and $C$ to decrease the total number of execution cycles for the same linear system. The experimental results also show that for the same combination of $F$ and $C$ an increase in $K$ may increase the execution cycles when solving the same problem.

![Figure 8: FPGA capacity and cluster effects, K=100](image)

![Figure 9: FPGA capacity and cluster effects, K=200](image)

### 4.6 System Scalability Issues

One of the first intuitions, when trying to speed up the execution of a problem with a given size $N$, is to increase the total number of execution units ($K$). In the experiment, the problem size is fixed at 1000 and $K$ is varied from 10 to 1200. The results are plotted in Figure 10, (a) total number of execution cycles, (b) execution rate per FPGA in GFLOPS, (c) total system execution rate in GFLOPS, and (d) the interconnect cycle percentage.

![Figure 10: System scalability analysis](image)

When the number of processing elements is less than around 15% of the matrix dimension $N$, the execution rate of the whole system scales relatively linearly with $K$. Then if $K$ is larger than 15% of $N$, the execution cycles are dominated now by the interconnect cycles, which is also shown in the sharp increase in the interconnect cycle percentage from 12% to 40%. Beyond that point, the execution time is only affected by interconnect-related parameters, such as the cluster size, link delay, etc. The sudden drop of execution rate when the value of $K$ is 640 is mainly due to the fact that the height of the 4-ary broadcast tree used increases from three to four levels. The fact that individual FPGA execution rate drops linearly with $K$ in the same region where the total execution rate increases linearly, shows that with a fixed problem size, the increase of total parallelism is accompanied by a linear increase in interconnect overhead, until the execution rate saturates and dominated by the interconnect overhead. Then the individual FPGA execution rate becomes inversely proportional to $K$, subjecting to the global interconnect effects, such as the broadcast latency, which increases coarsely with the total number of chips in the system and is dependent upon the broadcast network topology.

### 4.7 Comparison with Existing HPC solutions

With the analysis results from previous sections, in this experiment, a TeraBEE system is virtually constructed and simulated for its performance to be compared with existing supercomputers. The TeraBEE has 6561 Xilinx XC2VP125 chips, which is the currently most advanced FPGA available commercially. Each FPGA has 125,136 LUTs,
10Mbits on-chip SRAM, 1040 user I/O pins, and 20 MGT transceivers, each capable of 2.5 Gbps duplex bandwidth. The emulation capacity can support up to 48 parallel processing units, and the large number of I/O pins can connect to 12 independent DDR400 DIMMs, each with separate 64-bit data bus, 13 bit address, and 4-bit control pins. At one Gbytes each DIMM, up to 12 Gbytes off-chip memories are available to each FPGA, and each DIMM is capable of 3.2 GBps data bandwidth, thus enabling all 48 processing units continuously to operate without any limitation from memory bandwidth constraints. The processing FPGA’s are interconnected on a 3-D torus, each connected directly to its size nearest neighbors with three MGT links per direction. When running the FPGA at the target system rate of 100MHz, the MGT links introduce a link delay of 12 cycles. The aggregation tree and the control tree are built using a 9-ary tree, directly connected from level to level with parallel LVDS I/O signals, thus it only has one-cycle delay between levels.

Table 2: Existing top 5 supercomputers & TeraBEE

<table>
<thead>
<tr>
<th>Rank</th>
<th>Computer</th>
<th>N-half</th>
<th>N-max</th>
<th>Proc</th>
<th>R-max</th>
<th>R-peak</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NEC Earth-Simulator</td>
<td>266240</td>
<td>1075200</td>
<td>5120</td>
<td>35860</td>
<td>40960</td>
</tr>
<tr>
<td>2</td>
<td>HP ASCI Q - AlphaServer SC ES45/1.25 GHz</td>
<td>126100</td>
<td>590000</td>
<td>4096</td>
<td>7727</td>
<td>10240</td>
</tr>
<tr>
<td>3</td>
<td>HP ASCI Q - AlphaServer SC ES45/1.25 GHz</td>
<td>126100</td>
<td>590000</td>
<td>4096</td>
<td>7727</td>
<td>10240</td>
</tr>
<tr>
<td>4</td>
<td>IBM ASCI White, SP Power3 375 MHz</td>
<td>179000</td>
<td>518096</td>
<td>8192</td>
<td>7226</td>
<td>12288</td>
</tr>
<tr>
<td>5</td>
<td>MCR Linux Cluster Xeon 2.4 GHz – Quadrics</td>
<td>90000</td>
<td>450000</td>
<td>2304</td>
<td>5694</td>
<td>11060</td>
</tr>
</tbody>
</table>

Table 2 lists the current (Nov. 2002) world top 5 supercomputers performance [27] when running the Linpack benchmark program, as well as the simulated results for TeraBEE. The maximum value of matrix size $N$ can be run on the TeraBEE is only limited by the off-chip DRAM capacity. When using all FPGA’s available in the system, matrix size $N$ can be as large as 4,408,992. Since the $K$ value of 314928 is only 7% of the $N$ value, the achieved performance of 46.9 TFLOPS at this $N$ value is not the maximum achievable if more memory capacity is used. At the peak execution rate, TeraBEE runs at 7.15 GFLOPS per FPGA, which is slightly higher than the seven GFLOPS per processor achieved by Earth-Simulator. Nevertheless, the maximum matrix size is over four times larger than that of the current No. one supercomputer, the NEC Earth-Simulator, mainly because of the larger memory capacity on the TeraBEE system.

5 Conclusion

As shown in previous sections, using an FPGA-based RCP for large matrix scientific computations is not only feasible, but also more efficient than many existing supercomputer solutions. With the progression of Moore’s law, silicon chips with regular structures, such as memory and FPGA’s will be able to fully take the advantage of the exponentially increasing capacity available on the silicon die, without the exponentially increased complexity of chip design for general-purpose processors. By pushing the architecture design decisions after the fabrication of the silicon chip, FPGA’s can embrace new fabrication technologies much sooner than traditional processors, and make it possible to overlap the system architecture design with the FPGA chip design. With the modularity and regularity of a RCP system, it is even possible to re-architect the system to the particular application well after the system modules are finished. Furthermore, by running the FPGA’s at 100’s of MHz rather than several GHz, the chips consumes much less power, thus enabling much denser system to be built, and reduce the inter-module connection delays with shorter wires, which translates into better system performance.

Despite the superior performance achieved by a RCP on the Linpack benchmark, the real performance advantages of a RCP are achieved on target applications that can take the advantage of stream like processing or near neighbor computations. Many applications in the domain of signal processing and finite element problems naturally exhibit the above characteristics. When used on a RCP, the algorithm can be spatially direct-mapped on the FPGA’s, and then stream the data through the system, hence even reduce the amount of memory needed, the percentage of memory access time in the overall execution, and maintain near peak performance throughout the duration of the execution. If a reduced floating-point representation or fix-point number operations can be used instead of the IEEE compliant single/double precision operations, a RCP can achieve orders of magnitude higher performance than even the fastest supercomputers. Fundamentally, the ability to tailor the execution to the target application and the freedom to trade between spatial and temporal parallelism are the reason why a RCP will become the future of high performance computing.
References