Hyper-Systolic Routing for SIMD Systems

Andreas Hoferichter
HLRZ, c/o IfH/DESY-Zeuthen, D-15738 Zeuthen, Germany

Thomas Lippert and Klaus Schilling
HLRZ, c/o Research Center Jülich, D-52425 Jülich, Germany

Paolo Palazzari
ENEA, C. R. Casaccia, Via Anguillarese 301, 00060 S. Maria di Galeria, Italy

Hubert Simma
INFN, University La Sapienzia, Piazalle Aldo Moro, Rome, Italy

We emulate a router for the distributed permutation problem on SIMD parallel systems without hardware routing facilities, exploiting nearest neighbor connectivity. The novel method is based on the hyper-systolic algorithm. We discuss its implementation on APE100/Quadrics and present bandwidth measurements.

1 Introduction

Massively parallel SIMD systems like the Italian supercomputer APE100/Quadrics[1] are very attractive for high performance computing applications as they offer a pathway to *Teracomputing* in the near future[2] at an affordable price. The 3-dimensional interprocessor network of the supercomputer APE100/Quadrics has been designed to carry out fast data movements in quantum chromodynamics calculations and, as a consequence, supports a next-neighbor connectivity exclusively. Unfortunately, a lack of more general routing facilities presents a hindrance to the implementation of many scientific problems on this powerful machine.

In this work we propose a novel emulation of routing for the distributed permutation problem on parallel SIMD computers with next-neighbor connectivity on an n-dimensional toroidal network. The router is thought to provide a substitute for a dynamically reconfigurable switch network, providing a systematic routing approach for systems without hardware routing support.

On SIMD machines with next-neighbor connectivity like Quadrics it can be a workaround to alleviate or even overcome the lack of routing. Our method needs a total number of interprocessor communincation operations of $O(p^{\frac{3}{2}})$ next-neighbor data hoppings to perform a complete permutation of data between the processors, with p being the number of processors. The routing procedure is based on the hyper-systolic algorithm[3, 4, 5, 6]. In the hyper-systolic communication mode, the physical machine is mapped on a 1-dimensional systolic ring structure[7], along which the data is communicated among next-neighbors in the ring or between neighbors at a distance of k processors, connected by a second set of fast communication lines (hyper-connections). In this manner one can avoid redundant communication operations that occur in many systolic applications.

After the presentation of the hyper-systolic router for SIMD systems we describe its implementation on APE100/Quadrics and present measurements of the "bandwidth" of a general routing step on the 32-node Quadrics Q4 and the 512-node Quadrics QH4.

As part of the AQUA^{1000} software library for APE/Quadrics the routines will be made available for the Quadrics users[8]. In the future, the router could be useful for APEmille[2], the European Teracomputer, and for the Columbia Teracomputer[9], both machines being under construction.

2 Hyper-Systolic Routing

We start from an implementation network that supports (the emulation of) a 1-dimensional ring connectivity, see Fig. 1. Fast access to next-neighbor processors and to processors in

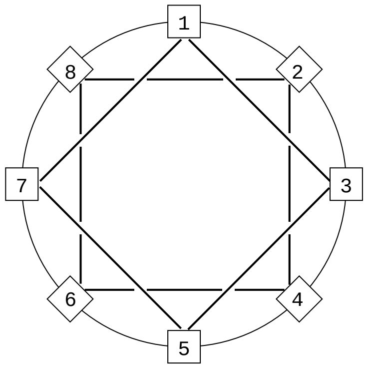

Figure 1: Systolic ring with $p = 8$ and $k = 2$.

distances 1 and k in an homogeneous data-parallel pattern is assumed along the ring.

To set the stage let us introduce a simple *systolic* routing emulator. We can choose the mapping $(x_1, x_2, \ldots, x_d) \to i$

$$i = \sum_{j=1}^{d} (x_j + 1) \prod_{\nu=0}^{j-1} n_\nu \tag{1}$$

to relate the processor coordinates $(x_1, x_2, \ldots, x_d)$. of a d-dimensional implementation network to a 1-dimensional systolic index i of the processors in the ring. n_ν is the number of processors in dimension ν, $0 \leq x_\nu \leq n_\nu - 1$ and $p = \prod_{\nu=0}^{d} n_\nu$ with $n_0 = 1$. In the same way we assign the coordinates of the destination processor to an index f_i that acts as a message header. The message array is a vector m_i of length M on each processor with index i, $i = 1, \ldots, p$. .

Algorithm 1 Systolic routing emulator.

```
foreach processor i = 1 : p ∈ systolic ring

  for j = 1 : M                    !! initialization
    m̃_i(j) = m_i(j)
  endfor
  m̃_i(M + 1) = f_i

  for l = 1 : p
    where m̃_i(M + 1) = i           !! load result array
      for j = 1 : M
        r_i(j) = m̃_i(j)
      endfor
    end where
    for j = 1 : M + 1               !! circular shift
      m̃_i(j) = m̃_{i mod_p +1}(j)
    end for
  end for

end foreach
```

For illustration, the pseudo-code of the simple systolic approach is given in Algorithm 1: first, the message m_i on processor i is copied to a travelling array $\tilde{m}_i$ of size $(M+1)$. The $(M+1)$th element of the moving vector carries the message header f_i. In the next step, it is decided locally if the header $\tilde{m}_i(M+1)$ at processor i corresponds to the index i and for true condition the travelling message is copied to the result array r_i. Then the travelling array $\tilde{m}_i$ is shifted by one processor along the ring to $\tilde{m}_{i+1}$ for all processors $i = 1, \ldots, p$ and again it is decided locally if the header $\tilde{m}_{i+1}(M+1)$ corresponds to $i+1$ and the moving vector is copied to the result array r_{i+1}. After p steps, the permutation of the array m_i into the array r_i is completed.

The communicational complexity of the systolic permutation is given by p moves of distance 1 along the systolic ring. In total, this amounts to a number $O(p^2)$ interprocessor communication operations. From the point of view of scalability, the systolic method decreases in efficiency with the processor number p and therefore will be not very useful on massively parallel systems[1].

In order to improve the scalability of routing on SIMD next-neighbor networks we next apply the *hyper-systolic* idea to the distributed permutation problem. In the hyper-systolic algorithm, an intrinsic redundancy of the interprocessor communication, as occurs in some systolic computations, is removed. This is achieved using intermediate systolic arrays. Hence, the hyper-systolic algorithm trades memory against speed. In the case of routing, however, memory is of minor importance.

We introduce hyper-systolic indices h_i and g_i on each processor i, $i = 1, \ldots, p$, that are given by

$$h_i = (i-1)\text{mod}_k + 1, \qquad g_i = (f_i - 1)\text{mod}_k + 1, \tag{2}$$

with k being the hyper-shift along the hyper-connections as indicated in Fig. 1.

The hyper-systolic router proceeds in two phases:

1. Initially, the message field m_i is assigned to a moving array $\tilde{m}_i$ of local size $M+2$. The *two* headers f_i and h_i are copied to addresses with index $M+1$ and $M+2$ respectively, as given in Algorithm 2.

   Where $g_i = \tilde{m}_i(M+2)$ is equal to h_i the message is copied to the $(\mu = 1)$th component of an intermediate systolic array vector t_i^μ, with $\mu = 1, \ldots, k$. This operation is followed by a shift of the moving array by 1 processor from location i to location $i+1$ along the systolic ring, and again the same condition is imposed, and where it is true, the moving array is copied to t_{i+1}^2. After k steps all intermediate arrays t_i^μ, $i = 1, \ldots, p$, are occupied such that the messages are at their final positions modulo k in the ring with respect to i.

2. In the second phase of hyper-systolic routing, the result array r_i is moved with $r_i(M+1) = i$. First for each μ $(\mu = 1, \ldots, k)$ the header $t_{i,\mu}(M+2)$ is compared to $r_i(M+1)$ and if the comparison is true, the corresponding entry is copied to the result array. Next the result array r_i is shifted by a hyper-shift of length k along the ring and again all stationary intermediate arrays t_μ are examined. After $\frac{p}{k}$ such steps the permutation problem is solved.

In total k shift operations by 1 processor and $\frac{p}{k}$ shift operations by k processors are needed. Choosing $k \approx \sqrt{p}$, the hyper-systolic router shows an improved scaling behavior compared to the systolic algrithm, as only a number $O(p^{\frac{1}{2}})$ shift operations must be carried out. This fact renders the new type of routing procedure useful on massively parallel machines with a

[1]The naive approach, broadcasting processor by processor and deciding in each step where the message has reached its destination, is by factors slower than the naive systolic approach.

large number of processing units.

Algorithm 2 Hyper-Systolic routing emulator.

```
foreach processor i = 1 : p ∈ systolic ring

   for j = 1 : M                          !! initialization
      m̃_i(j) = m_i(j)
   end for
   m̃_i(M + 1) = f_i
   m̃_i(M + 2) = g_i
   r_i(M + 1) = i

   for l = 1 : k
      where m̃_i(M + 2) = h_i              !! load intermediate arrays
         for j = 1 : M + 1
            t_{i,l}(j) = m̃_i(j)
         endfor
      end where
      for j = 1 : M + 2                    !! circular shift by 1
         m̃_i(j) = m̃_{i mod_p +1}(j)
      end for
   end for

   for l = 1 : p/k
      for μ = 1 : k                        !! load result array
         where t_{i,μ}(M + 2) = r_i(M + 1)
            for j = 1 : M
               r_i(j) = t_{i,μ}(j)
            endfor
         end where
      end for
      for j = 1 : M + 1                    !! circular shift by k
         r_i(j) = r_{i+k−1 mod_p +1}(j)
      end for
   end for
end foreach
```

3 Implementation on APE100/Quadrics

The APE100/Quadrics parallel system has been designed for efficient multiplications of 3×3 matrices and 3-vectors on large regular grids as required in quantum chromodynamics applications. The diagram Fig. 2 depicts the Quadrics system: compilation and program sub-

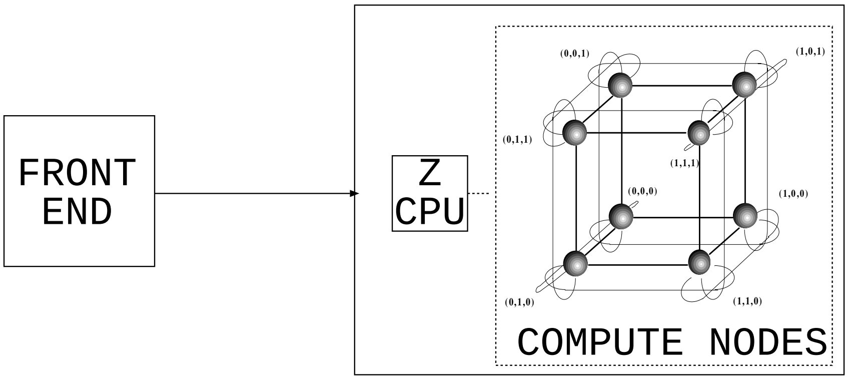

Figure 2: The Quadrics system.

mission as well as I/O tasks are carried out on a front-end workstation. The so-called Z-CPU manages the instruction stream (synchronous for all computing nodes) and performs global integer arithmetics.

The programming language TAO is based on so-called syntagmas, a kind of structs which are very useful for efficient programming in rather general applications also. TAO appears FORTRAN- and C-like, primarily supporting a local address space whereas communication from and to remote memories uses additional global address bits.

The compute nodes perform the local floating point operations. They each are composed of memory, a floating point unit called MAD, and the communication chips. The MAD is able to perform 2 floating point operations within one clock cycle, that is $(25\,\mathrm{MHz})^{-1}$. Thus, one processor can perform with 50 Mflops.

The connectivity. The smallest configuration of the Quadrics computers is a Q1, a unit of 8 computing nodes, called a board. The nodes are connected by 6 communication links to their neighbor nodes in a 3-dimensional grid, see Fig. 2 to illustrate the network topology. Larger machines like the 32-node Q4 or the 512-node QH4 are composed of several boards. The network topology of the large system is hard-wired. A 512-node Quadrics QH4 is typically configured as an $8 \times 8 \times 8$-grid, toroidally closed in each direction.

In order to transfer one 32-bit word from the memory of the neighboring processor to the memory of the local processor, apart from start-up times, 8 clock cycles are needed in the ideal case. In practical application this limit can not be reached completely, however.

3.1 Emulating a systolic ring on Quadrics.

The 3-dimensional network of the Quadrics can be exploited to emulate a 1-dimensional ring functionality. The higher dimensionality of the implementation network provides the required hyper-connections along the virtual ring.

In Fig. 3a, we sketch the assignment of the processors onto a systolic ring (of 32 processors) and demonstrate how we can perform a shift by one processor along the ring carrying out three hardware communication operations. In Fig. 4, a 'hyper-shift' by 4 processors is

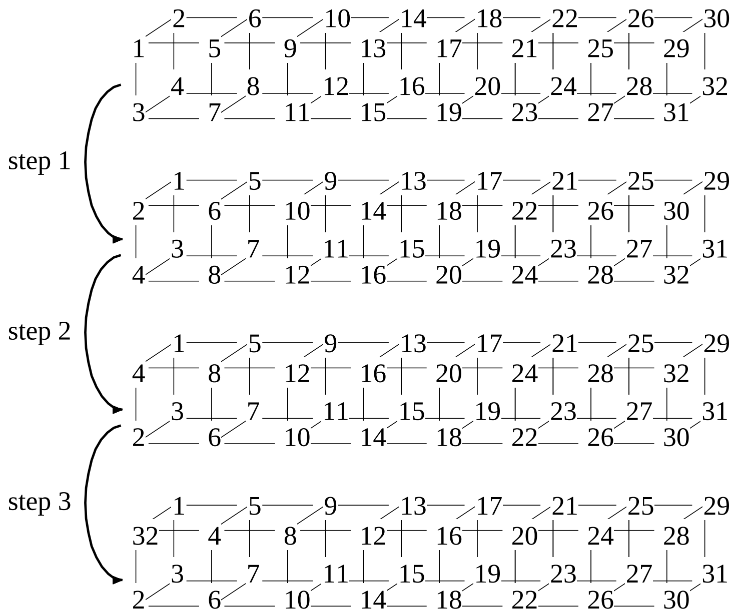

Figure 3: Shift of data to the next-neighbor processor on the systolic ring emulation.

depicted. Here only 1 hardware communication step is required.

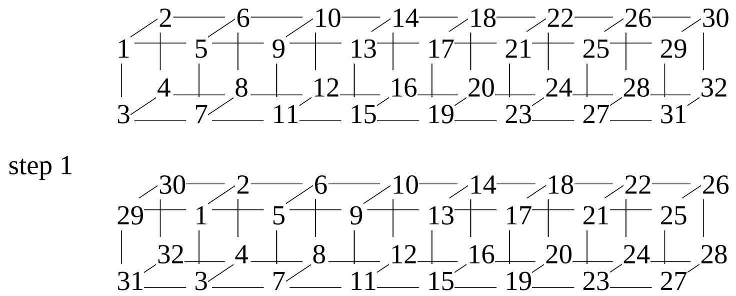

Figure 4: Hyper-shift of data by 4 processors on the systolic ring emulation.

3.2 Measurements of Bandwidth

We measured the bandwidth of a complete permutation in terms of the communication time for the transfer of one word to a neighbor processor. We have investigated the performance of the systolic and the hyper-systolic version for message lengths ranging from $n = 1$ to 2000. Particular effort has been made to ensure efficient memory to register transfer rates, by application of blocking of the message vectors into blocks that are adapted to the register size of 128 words.

In Fig. 5a, we depict the times that are required to perform a complete routing step on the 32-node Quadrics Q4. For small message length, the influence of the start-up time can be observed. This time is partly due to the message header consisting of 2 words.

We introduce a complexity model for interprocessor communication. From an analysis of the assembler code, we estimate the interprocessor communication start up time t_l for our code on APE100/Quadrics–averaged over the whole shift process–to be bout 10 times as large as the sustained time t_m per communicated word. Moreover, the two headers have to be counted. The communication time is then given as

$$T(M) = t_l + (M + 2)t_m, \tag{3}$$

with $t_l = 10t_m$. The line in Fig. 5a is drawn to compare with the theoretical prediction from the communication model[2].

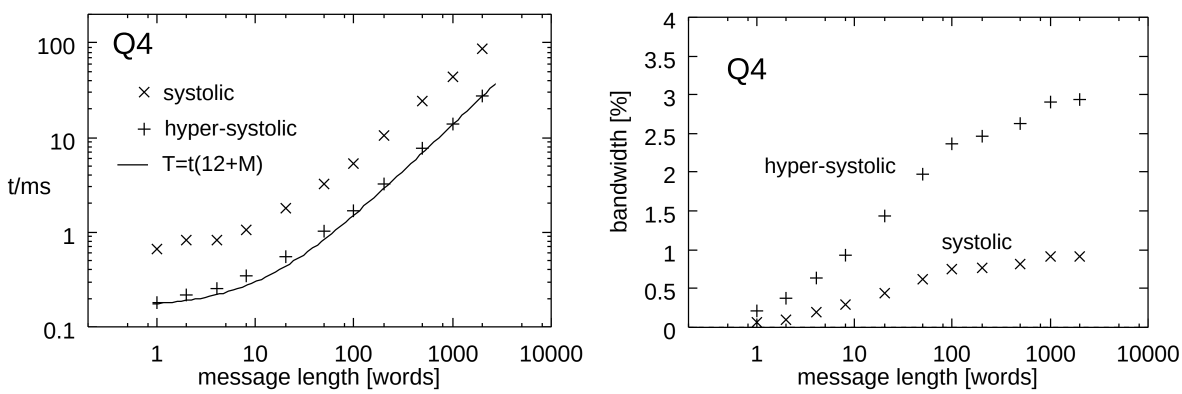

Figure 5: Times of systolic and hyper-systolic solution of the distributed permutation problem on Quadrics Q4 and bandwidth in % of one next-neighbor communication step.

[2]The assembler code optimizer on APE100?Quadrics tries to overlap communication and computation processes. The resulting code is rather involved and therefore, the real start-up times and sustained times can only be given on average.

Fig. 5b gives the "bandwidth" in terms of the communication time of 1 word to a next-neighbor processor. The complete permutation on Q4 is about 30 times slower as the underlying next-neighbor communication step.

Ideally, the hyper-systolic algorithm's interprocessor comunication expense should scale according to $p^{\frac{1}{2}}$. The optimal scaling behavior depends on the cost function, *i. e.*, the number of hardware communication steps for a given cshift of length k along the systolic ring[5].

On a 512-node QH4, the optimal shift constant, for the cost function given, is $k = 16$. This shift is carried out on the back of 3 hardware communication operations, however, in contrast to 1 operation on Q4. The reulting times for permutation therefore increase by a factor of about 10 instead of about 6 as given by the ideal scaling law, for the full permutation.

Fig. 6 shows the bandwidth together with the measured times.

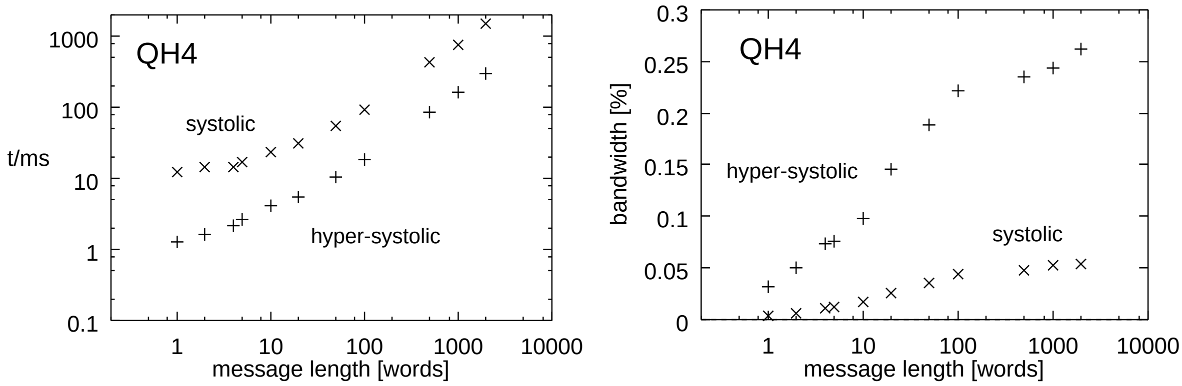

Figure 6: Times of systolic and hyper-systolic solution of the distributed permutation problem on the 512-node Quadrics QH4 and bandwidth in % of one next-neighbor communication step.

Acknowledgments

The tests in this investigation have been carried out on the Quadrics Q4 at HLRZ/Jülich and the Quadrics QH4 of the HPCN-group at ENEA/Casaccia, Italy. Th. L. thanks N. Petkov for important discussions. The work of Th. L. is supported by the Deutsche Forschungsgemeinschaft DFG under grant No. Schi 257/5-1.

References

[1] C. Battista et al.: 'The APE-100 Computer: (I) the Architecture', Int. J. of High Speed Computing 5 (1993) 637.

[2] R. Tripiccione, in: F. Karsch et al. (edts.), Proceedings of the International Conference "Multi-scale Phenomena and their Simulation", ZiF, Bielefeld, Sep. 30 - Oct. 4, 1996, to appear.

[3] Th. Lippert, A. Seyfried, A. Bode, K. Schilling: 'Hyper-Systolic Parallel Computing', Preprint Server HEP-LAT 9507021, Preprint WUB 95-13, HLRZ 32/95, submitted to IEEE Trans. of Parallel and Distributed Systems.

[4] Th. Lippert, U. Glässner, H. Hoeber, G. Ritzenhöfer, K. Schilling and A. Seyfried Int. J. of Mod. Phys. C 7 (1996) 485-501.

[5] P. Palazzari, Th. Lippert and K. Schilling: ‘Simulated Annealing Techniques for the Communication-Efficient Hyper-Systolic Parallel Computing on APE100/Quadrics’, in: L. Grandinetti et al. (edts.), Proceedings of NATO Advanced Research Workshop on High Performance Computing, Technology and Applications Cetraro, Italy - June 1996 (NATO ASI Series, Kluwer, 1996).

[6] Th. Lippert, K. Schilling: ‘Hyper-Systolic Matrix Multiplication’, in: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA’96, August 9-11, 1996, Sunnyvale, California, Hamid R. Arabnia (edt.) (C.S.R.E.A. Press, 1996, p. 919).

[7] N. Petkov: *Systolic Parallel Processing*, (Amsterdam: North-Holland, 1992).

[8] The software and related publications are available under (http://www.hlrz.kfa-juelich.de/HLRZ/HighEnergy/AQUA)

[9] I. Arsenin et al. in: T. D Kieu et al. (edts.), Lattice 95, Proceedings of the International Symposium on Lattice Field Theory, Melbourne, Australia, 1995, Nucl. Phys. **B** (Proc. Suppl.) 47 (1996) 804.
N. Christ, in: F. Karsch et al. (edts.), Proceedings of the International Conference ”Multiscale Phenomena and their Simulation”, ZiF, Bielefeld, Sep. 30 - Oct. 4, 1996, to appear.