# An adaptive, reconfigurable interconnect for computational clusters

A Shafarenko
Department of Computer Science, University of Hertfordshire,
Hatfield, AL10 9AB UK
a.shafarenko@computer.org

V Vasekin
Telecom MODUS Limited, Leatherhead, Surrey, KT22 7SA, UK
vladimir.vasekin@t-modus.nec.co.uk

## Abstract

*This paper describes the principles of an original adaptive interconnect for a computational cluster. Torus topology (2d or 3d) is used as a basis but nodes are allowed to effectively migrate along the torus cycles. An optoelectronic scheme which makes such migrations possible with only local synchronisation is outlined. Between the instances of migration the interconnect behaves as a direct packet-routing network which constantly monitors its traffic parameters. A decentralised predictive algorithm is applied periodically to decide whether the current topology is consistent with the predominant traffic flow and if it is not, a reconfiguration to a better-matched topology occurs. We present simulation results that show that on some standard computational benchmarks a significant speedup is possible as a result of automatic matching between the effective topology of the application's message-passing infrastructure and that of the interconnect.*

## 1 The case for reconfiguration

In this paper we shall present an interconnect scheme which demonstrates a high-degree of adaptivity owing to its capability to reconfigure the topology of the network in sympathy with the prevailing traffic patterns. This section will briefly introduce certain well-known trade-offs in cluster interconnects which motivate our work.

**Bus vs packet routing.** Two forms of interconnect are used in cluster systems: a bus (or a set of connected buses), for example fast Ethernet, and a packet-routing network which is a set of bidirectional channels connecting routing nodes (routers). The cluster machines interface with the buses (in a bus system) or routers (in a packet-routing system) in order to send and receive messages. An interconnect is called (dynamically) reconfigurable when connections between buses or channels can be broken and reformed repeatedly in the course of program execution.

On a bus, a message is physically delivered to every connected node and the node itself determines whether or not to receive the message. In a packet-routing system, messages are forwarded along certain routes to their final destinations. The advantage of a packet-routing network over a set of buses is subtle. It has not been discussed in detail in available literature, but certain fundamental differences are easy to see. For example, in the case of uniform random traffic (URT) under a heavy load, a mesh of buses loses to a 2d packet-routing torus a factor of 4 in throughput. This is due to the $N/4$ hops a message has to make on average along one dimension of a torus versus the effective $N$ "hops" resulting from bus reservation for one message transaction. However, this difference is a constant factor independent of system size. In reality, the bus slows down as its physical size increases (which is what is usually meant by the lack of true scalability) to compensate for the increase in the required transmitter power. Nevertheless, a free-space *optical* bus can be built that does not degrade appreciably in the whole practical range of clusters (1-100 nodes). Under a moderate load, when the throughput is not a concern, a bus interconnect could show a much better latency thanks to its ability to deliver messages directly to their destinations.

**The role of locality.** A packet-routing network comes into its own when the application affords a large degree of *communication locality*. If the algorithm-induced traffic and the network topology are such that messages make an average of $n$ hops per dimension before they reach their destinations, a packet-routing torus beats the equivalent bus mesh by a factor of $N/n$, both in throughput and latency. For a fixed $n$, this factor gives the packet-routing network a *scal-*

*able* advantage over the buses, which may be the reason why they are used in massively-parallel computers, such as the Cray T3E. Certainly, at least for a large cluster packet, routing should be attractive.

The ability to exploit locality, important as it may be, is predicated on the matching of the "natural connectivity" of an application with the topology of the network. At present this means that the application code must take topology into account explicitly, i.e. by aligning the data and processes with the network structure. Such an approach is undesirable in data-parallel systems where an abstract model of array processing is used to avoid targeting a particular architecture. Furthermore, in real-life applications task-parallelism (whereby several data-parallel operations can be performed concurrently by different cluster nodes) is just as important. Here a large degree of communication locality is still preserved, as every array element interacts with only a small set of other elements; these elements, however, may form a neighbourhood which no longer conforms to the hardware locality induced by the network topology. Process migration and data re-mapping on a fixed topology are possible solutions, albeit requiring processes to be light-weight and mobile, which is difficult to achieve.

**Reconfigurability.** The above analysis seems to suggest that packet-routing networks for generic cluster computing require reconfigurability to ensure that their advantage in latency is fully exploited. There are, however, very few examples of such interconnects in literature. Some of them, such as the RMN system [5], use a fixed set of configurations to enable a number of $2d$-mesh-based and $n$-cube-connected algorithms to exploit their locality. Another example of reconfigurability is the Achilles system[13] which boasts extremely low latency, which is achieved at the price of having to support full connectivity across a single $8 \times 8$ crossbar switch.Unlike RMN, the Achilles switch is controlled by the headers of incoming packets and is, consequently, decentralised. However, the complexity of this solution and its lack of scalability prevents it from being used in larger clusters.

The bulk of literature on reconfiguration technology is concentrated in the area of reconfigurable connections on programmable logic chips; see, for instance,[11, 10]. Optical communication is appreciated as a technology that supports the massive connectivity that large distributed systems need[9, 4]. Scalable interconnects for cluster computing have started to appear[9]. However, even in the largest optically-connected systems being discussed (see, for example, a survey in [7]), reconfigurability has not yet been utilised, despite the fact that theoretical advantages of optical reconfigurability are well-understood [8, 2].

On a systems level, it is our position that a generic reconfigurable interconnect requires both channel-switching and message-routing. Indeed, in any given configuration, there is no static guarantee that the application will only use direct connections between nodes. The topology usually follows the prevailing connectivity of the application but not *all* its communication requirements. Forwarding packets to their indirect destinations is still necessary even though a channel could be switched to provide direct access. The problem is that channel switching is much slower than packet routing, even for very fast channels, such as optical ones. The delay due to channel switching can only be justified by the savings in forwarding costs of a long train of messages, i.e. a prevailing traffic pattern.

In this paper we propose a novel approach to reconfiguration which enables an optoelectronic packet-routing network to reconfigure automatically and in a decentralised manner. This is physically achieved by exchanging optical beams that connect a pair of nodes to cause the nodes to effectively swap their positions in a network cycle. Holographic beam-steerers are a mature technology which can be used for reconfiguration purposes (see [14]) and controlled electronically. At the nodes, optically-transmitted packets are converted to electrical signals and either consumed or routed electronically to one of the outputs for a subsequent optical transmission.

The decision to perform a beam exchange is made locally, at a node, by observing the traffic flow through the nodes and predicting the effect of reconfiguration by a heuristic rule. This approach avoids the use of a single cross-bar switch and also benefits from the fact that the capability to deflect an optical beam (unlike the capability to switch an electrical wire) imposes no propagation-time penalty by itself.

After several exchanges a distant migration of any node can occur, which, for a given node, is only restricted by the limits of its network cycles. The scheme supports a very large number of potential configurations and is totally transparent to the user code.

## 2 Reconfigurable torus

A torus network is the Cartesian product of $d$ primitive bidirectional cycles. Our proposal achieves long-distance network alterations as a composition of several per-cycle reshuffles. It should be noted that, since we do not require global synchronisation, those alterations could be performed in parallel with each other and the ongoing computation processes. In order to introduce our solution, let us consider the primitive cycle in Fig 1. The figure shows the traffic flowing in one direction (clockwise). The part of the traffic traveling anticlockwise is completely independent. At some stage in the computation, the node *b* may detect that it is forwarding too much transit traffic from *a* to *c* which prevents *b* from injecting its own messages clock-

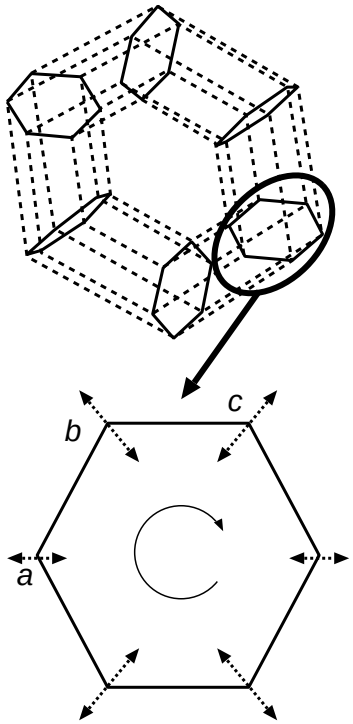


**Figure 1. Single cycle of a reconfigurable network**

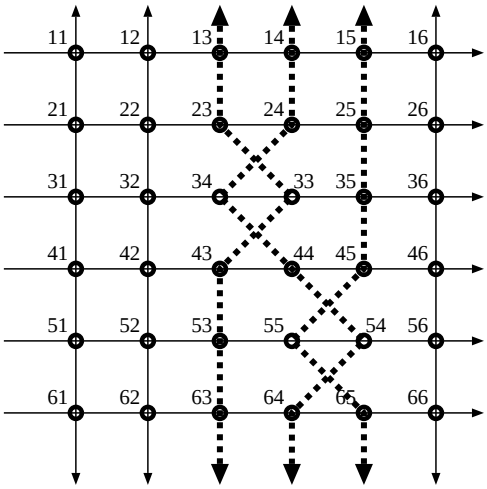


**Figure 2. Torus topology after exchanging pairs (33,34) and (54,55)**

wise at the required rate. Its own messages include those that come to it from the other dimension(s) (they are shown as double-headed arrows in the figure for the 2-dimensional case). If the node *b* can exchange places with the node *a* in the clockwise cycle, that would reduce the amount of transit traffic by shortening the effective distances for the passing messages.

Interestingly, such a reconfiguration can be performed using only local synchronisation. i.e. one involving nodes *a*, *b* and *c*. Moreover, by combining several exchanges, a node can be forced to migrate as far as necessary around one or more cycles. Since a migration in one cycle can never result in two distinct nodes becoming the same, the guaranteed minimum distance for a pair of nodes drifting towards each other is equal to the number of independent dimensions. As a result of nodal exchanges in the orthogonal dimension(s), the cycles effectively bend and twist, see the example in fig 2, where three vertical cycles: 3,4, and 5, have been affected by two horizontal exchanges. Still, under no circumstances does this lead to a cycle being broken and re-formed with a different set of nodes included in it. Consequently, the proposed scheme is not an arbitrary permutation of the network order and hence may not support a globally optimal solution. However the variety of configurations this restricted scheme does support is enormous and so it is likely that efficient ones occur among them. It should also be noted that, as reconfiguration proceeds independently in clockwise and anticlockwise components of a cycle, the same node can migrate to different places in them. Depending on the direction of incoming traffic, every node is in fact a combination of $2d$ "ghosts", e.g. there are four ghosts for each node in a 2-dimensional torus: (East→West, North→South), (East→West, South→North), (West→East, North→South), and (West→East, South→North).

Let us proceed to the proposed decentralised reconfiguration algorithm. Consider a pair of adjacent nodes on the same cycle, say the clockwise one, with the message fluxes as per Fig 4.

Messages coming to the node $a$ constitute the flux $P_{in}$. Some of these messages get absorbed at that node (including those that are to be routed into the other dimension), with the rest, $P_{trans}$, being sent to $b$. At the same time some messages are originated at $a$, giving rise to the flux $Q_{trans}$. These include messages coming in to $a$ from the other dimension. At $b$, some messages from $P_{trans}$ and $Q_{trans}$ are absorbed and new ones are originated. These result in the fluxes $P_{out}$, $Q_{out}$ and $R_{out}$, respectively.

The decision on whether $a$ and $b$ should swap depends on how many messages per second would have their paths extended and how many reduced. The difference of those numbers, M, is as follows:

$$M = (P_{trans} - P_{out}) - (P_{in} - P_{trans}) + Q_{out} \\ -R_{out} - G(Q_{trans} - Q_{out} - F_{ba})$$

The first term is the incoming flux absorbed at $b$. Those messages need to reach $b$ and have to make an extra hop over $a$ in order to do so. The second term is the number of messages per second that are absorbed at the node $a$ and which would gain a hop as a result of the swap. The negative sign accounts for this gain. The third term stands for the traffic generated at $a$ to be dispatched beyond the node $b$ and hence it would lose a hop as a result. The term describes the traffic originated at $b$ gaining an extra hop. Finally, some messages originated at $a$ are intended for $b$. As a result of the swap they would have to follow a different route, as $b$ would be behind $a$ and thus unreachable in this cycle other than by a very long route. This circumstance is accounted for by the last term which contains two parameters. The value of $G$ is the number of additional hops for the messages transferred to the anticlockwise flow. Parameter $F_{ba}$ is introduced to account for the flux from $b$ to $a$ in the anticlockwise flow which is not shown in the figure. The fluxes have been observed over a period of time $T$. Now assume that all fluxes are constant. Then a positive $M$ has the meaning of the number of hops that the messages would have saved had $a$ and $b$ been swapped in the beginning of the period. Assuming that the fluxes vary slowly over the period $T$ and beyond, the value of $M$ could serve as a predictive criterion for reconfiguration. The reconfiguration cost normalised by $T$ should be subtracted from $M$ to work out the likely net effect of the new topology for the next period of observation .

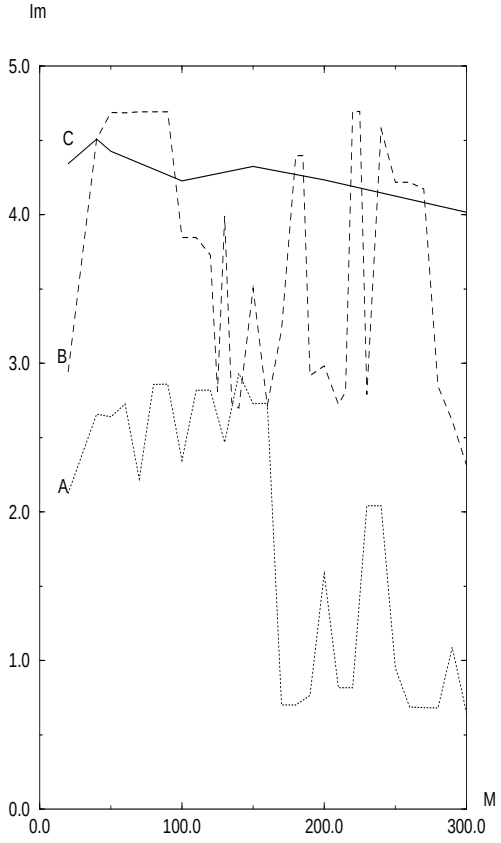


**Figure 3. Normalised performance $I_m$ vs threshold $R$**

## 3 Performance evaluation

The performance of the reconfiguration scheme was evaluated by simulating a reconfigurable network of processors running a curtailed version of an application benchmark.

**Simulated application.** In order to assess the above reconfiguration scheme, an established computational benchmark is required. Although benchmarking is prone to bias as the choice of a "typical" algorithm is bound to be subjective, it is still possible to obtain qualitative results that show the effect of the proposed scheme. For the purposes of demonstrating the adaptivity of our interconnect, we selected three classical Livermore Loops[12]: Kernels 7, 18 and 21, see fig5. Kernel 7 represents a one-dimensional finite-difference method with index locality 7, i.e. 7 consecutive neighbours of each element form its local neighbourhood (in the index, rather than network location, sense). Kernel 18 is a hydrodynamic piece operating in 2d on a $3 \times 3$ index template (i.e. locality 9) and finally Kernel 32 is a straightforward implementation of matrix dot product with nonlocal reductions.

**Parallelisation and node abstraction.** As the index pattern of all three Kernels is fixed there was no need to include the floating-point arithmetic, or indeed any calculation at all, in order to simulate the load that the processor presents to the interconnect. We assumed that the application was communication-bound, and thus neglected the processing time between the delivery of the inputs to an assignment and the emission of its write request. This assumption is justified in the fine-grain limit, where the intention is to extract as much parallelism as possible. For a courser-grain application, where communication is not as critical, the ad-

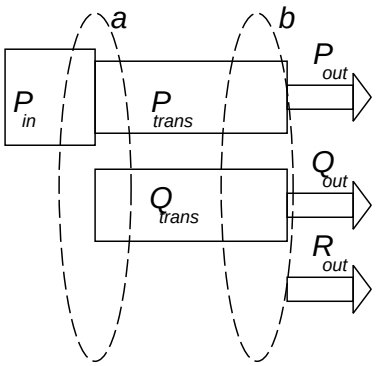


**Figure 4. Analysis of data streams through a pair of nodes**

vantage of a reconfigurable interconnect may be less pronounced.

In order to achieve significant network load, each node ran 8 parallel threads executing consecutive iterations of the innermost loop within its share of the iteration space. Due to the data-parallelism of the Kernels, those iterations were strictly independent. The multithreaded execution is necessary to account for the node capability to tolerate latency, without which fine-grain distributed computing would be impossible even with a sufficient network bandwidth. The introduction of roughly as many threads as there are independent channels incident to the node creates enough concurrency to prevent that, see [3, 1].

**Network abstraction.** For these experiments we chose the 3d torus topology because of its smaller diameter and hence more pessimistic expectations of the possible benefits of reconfiguration. For the sake of simplicity, our network simulation was synchronous, i.e. operating in lockstep. At each step, a node can insert a packet in the router and/or receive a packet from it. At every node, each of the 14 channels (6 pairs of links plus the node injection/absorption channels) has the capability to transfer one packet per step provided that a packet is available at the source and there is a free buffer at the destination. Buffering is set to 32 packet-size buffers on either side of each link. It should be noted that as optical technology enables very wide links (100 bits and more, due to high density of on-chip optical interface and/or the need to demultiplex fast optical channels), it makes sense to simulate packet-wide channels as this simplifies both routing and reconfiguration as well as improving the overall throughput and latency of the interconnect.

The following types of packets were used:

1. read request, containing a virtual memory address
2. data, containing the data read from memory in response to a read request
3. write request, containing a virtual memory address and its new content
4. write acknowledgement
5. system control packet, containing control data

Each packet carries its source and destination addresses and the thread ID. System control packets are used for notifying configuration events to the routers. Dimension-order routing was simulated, whereby a packet coming to a transit node is transferred to the next node en-route, first in the $x$ dimension, then in $y$ and finally in $z$. The transfer occurs only when there is a free buffer associated with the output link.

**Implementation of reconfiguration.** This was put into effect in line with the algorithm presented in the previous section. Since the statistic gathering required for the formula is local, a pair of consecutive nodes only require a low-throughput connection to exchange the statistical data from time to time. Compared to the main data traffic this communication is negligible and hence need not be simulated. The only thing that requires simulation is the process of position change which takes place when the decision to reconfigure has been made on the basis of the formula. This is achieved by blocking the node $a$ input (see fig2) for $\tau_r$ steps; switching the channels so that $b$ links up with $a$ and then with $c$; and then unblocking the $a$ input which should now link up with $b$. The reconfiguration time $\tau_r$ was set to 32 steps to reflect the current contrast between the speed of the optical link (of the order of 10 packets per $\mu$s) and the optical switch time (between 1 and 10$\mu$s). At the next step a system control message is broadcast around the network cycle to notify the other nodes of the node swap[1]. Special care was taken to avoid reconfiguration-induced deadlocks: a packet, once emitted is not permitted to change its direction en-route even though the updated routing tables may suggest otherwise.

**Data mapping and load balancing.** Normally array data would be mapped on a conforming processor arrangement to benefit from their inherent locality of access. As we were deliberately using a mismatched 3d arrangement of processors to challenge the adaptive properties of our interconnect, we chose to apply a scatter mapping as follows.

[1]It should be noted that in our decentralised reconfi guration methodology there is no need to block the whole cycle until the system broadcast message has been received by every node it is intended for. Indeed, a delay in recognising the new topology by any nodes only leads to suboptimal routing (say clockwise by 6 hops rather than anticlockwise by 5)

First of all, a global address space was introduced with virtual addresses in a continuous range from 0 to $L$. A cluster node with the coordinates $(x, y, z)$ was then assigned all virtual addresses $a$ such that $a \bmod N^3 = xN^2 + yN + z$, where $x$, $y$ and $z$ range from 0 to $N-1$ with $N$ being the period of the torus. This is essentially a 1d, cyclic distribution of addresses over a 3d interconnect structure irrespective of the data-array rank and shape. The data arrays manipulated by the Kernels were allocated one after another starting with virtual address 0 without any alignment gaps between objects. The "computational load", i.e., in our case, responsibility to issue network packets, has been divided between the processors by splitting the iteration space of the innermost loop into chunks of equal size and assigning one chunk to each node of the system in the order of their virtual addresses.

**Simulation results.** For simulation purposes, the three Kernels mentioned above were enclosed in a single loop which was iterated long enough to collect reliable statistics.

The reconfiguration criterion from section 2 was used to determine when to reconfigure local connections at every node. The results are shown in fig 3 (curve A) which plots the speed-up associated with reconfiguration against the threshold value for $M$, denoted as $R$. It is clear that a high threshold causes reconfiguration to occur too infrequently which weakens its effect. Since there is a cost associated with topology alteration and the prediction of the fluxes is never fully accurate, the benefits fail to materialise. In fact there is a net loss at some values of R. On the other hand, when the threshold is too low the reconfiguration cost is larger than the benefits. Consequently, useful behaviour can be expected at intermediate values of $R$, which is, in fact, what is observed. The troughs on the curve are due to the interplay of numerical parameters of the simulated application (such as array size), which can be smoothed out by randomising the parameter $T$ (curve B) and almost totally eradicated by making the threshold $R$ dependent on the system behaviour (curve C). In the last case $R$ represents the starting value of the threshold which varies in time depending on the actual performance: up when the nodal exchanges are too frequent and down when they are too rare. Under such self-adjustment, the effect of reconfiguration reaches high values (a factor of 4 in this benchmark).

**Related work.** Afsahi and Dimopoulous[2] support our assumption that structured communication locality is present in most cluster computing applications. Like ourselves, they propose to monitor traffic locally to predict communication patterns. The main difference between their solution and the one presented here is in the interconnect being controlled. Paper [2] is based on the *channel-switching* model where each node has a fixed number of retargetable connections, and so the question is, which one of them should be switched when an output packet is to be sent to a different destination. The intention is to keep the connections that are likely to carry the majority of packets as steady as possible in order to minimise their switching overheads. Afsahi and Dimopoulous proposed and evaluated various heuristics, such as LRU, LFU, etc., which are similar to the well-known virtual-memory paging heuristics. In the present paper we use *packet routing* for all packets so the links need not be switched just to deliver all packets. As a consequence, our solution could utilise switches whose switching time far exceeds the inverse throughput of the link — and that is the case in most optical networks. The slowness of the switch in our case only leads to diminished adaptivity.

Also note the use of machine learning in predicting shared-memory traffic in multiprocessors[6]. Techniques such as these could be used in reconfigurable routing networks in future.

## Conclusions

Optical communication technologies enable new approaches to a cluster interconnect. An adaptive toroidal interconnect has been proposed and its capability to automatically adapt to the application-induced traffic has been demonstrated.

Further research may address the following issues.

1. Optimal navigation across a variable-topology network. In a reconfigurable network the calculation of the shortest routes is an NP-complete task. There is a trade-off between the complexity of the decision making in the navigation logic of a router, the amount of global information it requires and the variety of available configurations. So far we have used the simplest solution, based on essentially fixed-network routing strategies, which have been modified to account for local alterations of the interconnect. Further research is required to address this problem in full.

2. The multicast problem. It is well-known that distributed algorithms exhibit complex multicast behaviour. While various forms of multicast can be emulated by point-to-point communications, the proposed optical solution can be modified to directly support multicast capability.

3. As reconfiguration in our approach is based on an optical switch, the issue of switch architecture is quite important. In particular, the number of different configurations that the switch needs to support should be kept to a minimum to reduce the cost and complexity of its control logic and optical elements. In this paper

we assume sufficient switching capability to support all permutations of a cycle. It is possible, however, that a proportion of cycle permutations can be left unsupported resulting in a cheaper switch without significant degradation of adaptivity.

## Acknowledgement

This research has been supported by an EU grant within the Framework IV MEL-ARI programme. The authors are grateful to Alex Bolychevsky for fruitful discussions and to our European partners whom we consulted on issues pertaining to optical component architecture.

## References

[1] A.Bolychevsky, C.Jesshope, and V.Muchnik. Dynamic scheduling in risc architectures. *IEE Proc.-Comput. Digit. Tech.*, 143(5):309–317, september 1996.

[2] A. Afsahi and N. J. Dimopoulos. Communications latency hiding techniques for a reconfigurable optical interconnect: benchmark studies. In *Applied Parallel Computing. Large Scale Scientific and Industrial Problems. 4th International Workshop, PARA'98. Proceedings.* Sprnger-Verlag, Berlin, Germany, 1998.

[3] D. B. Barsky and A. V. Shafarenko. Uniform random traffic in a massively-parallel data-drive n computer. In *Proceedings of MPCS'96*, pages 546–553. IEEE, 1996.

[4] C. Berger et al. Comparison of two reconfigurable $n \times n$ interconnects for a recurrent neural network. *Optical Review*, 3(6A):388–390, 1996.

[5] S. M. Bhandarkar and H. R. Arabnia. Parallel computer vision on a reconfigurable multiprocessor network. *IEEE Transactions on Parallel and Distributed Systems*, 8(3):292–308, Mar. 1997.

[6] M. F. S. et al. Predicting multiprocessor memory access pattern with learning models. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 305–312, 1997.

[7] J. Gourlay, T. Yang, J. A. B. Dines, J. F. Snowdon, and A. C. Walker. Development of free-space digital optics in computing. *Computer*, 31(2):38–44, 1998.

[8] M. W. Haney and M. P. Christensen. Fundamental geometric advantages of free-space optical interconnect. In *Proceedings of the Third International Conference on Massively-Parallel Processing Using Optical Interconnections*, pages 16–23, 1996.

[9] J.Dines, J.Snowdon, M.Desmulliez, D.Barsky, A.Shafarenko, and C.Jesshope. Optical interconnectivity in a scalable data-parallel system. *Journal of Parallel and Distributed Computing*, 41:120–130, November 1997.

[10] L. K. John and E. John. A dynamically reconfigurable interconnect for array processors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems. vol.6, no.1; March 1998; p.150-7*, 6(1):150–157, March 1998.

[11] L. Kurian and E. B. John. Development of free-space digital optics in computing. *IEEE Transactions on VLSI*, pages 150–157, March 1998.

[12] F. H. McMahon. The livermore fortran kernels test of the numerical performance rangemance. In J. L. Martin, editor, *Performance Evaluation of Supercomputers*, pages 143–186. North Holland, Amsterdam, 1988.

[13] J. Morris and S. Tham. High bandwidth and low latency from a three-dimensional reconfigurable interconnect. In *Proceedings of the SPIE The International Society for Optical Engineering*, volume 3844, pages 43–48, 1999.

[14] Suning-Tang and R. T. Chen. Reconfigurable electro-optic interconnects using holographic elements. In *Proceedings of the SPIE The International Society for Optical Engineering.*, volume 3288, pages 153–163, 1998.

```
c*******************************************
c***  KERNEL 7      EQUATION OF STATE FRAGMENT
c*******************************************
 1007 DO 7 k= 1,n
        X(k)=     U(k  ) + R*( Z(k  ) + R*Y(k  )) +
     1        T*( U(k+3) + R*( U(k+2) + R*U(k+1)) +
     2        T*( U(k+6) + Q*( U(k+5) + Q*U(k+4))))
    7 CONTINUE

c*****************************************************
c***  KERNEL 18     2-D EXPLICIT HYDRODYNAMICS FRAGMENT
c*****************************************************
 1018    T= 0.003700d0
         S= 0.004100d0
         KN= 6
         JN= n
         DO 70  k= 2,KN
         DO 70  j= 2,JN
          ZA(j,k)= (ZP(j-1,k+1)+ZQ(j-1,k+1)-ZP(j-1,k)
     1    -ZQ(j-1,k))*(ZR(j,k)+ZR(j-1,k))/(ZM(j-1,k)
     2    +ZM(j-1,k+1))
          ZB(j,k)=(ZP(j-1,k)+ZQ(j-1,k)-ZP(j,k)-ZQ(j,k))
     1         *(ZR(j,k)+ZR(j,k-1))/(ZM(j,k)+ZM(j-1,k))
   70    CONTINUE
c
         DO 72  k= 2,KN
         DO 72  j= 2,JN
         ZU(j,k)=ZU(j,k)+S*(ZA(j,k)*(ZZ(j,k)-ZZ(j+1,k))
     1                 -ZA(j-1,k) *(ZZ(j,k)-ZZ(j-1,k))
     2                 -ZB(j,k)   *(ZZ(j,k)-ZZ(j,k-1))
     3                +ZB(j,k+1) *(ZZ(j,k)-ZZ(j,k+1)))
         ZV(j,k)=ZV(j,k)+S*(ZA(j,k)*(ZR(j,k)-ZR(j+1,k))
     1                 -ZA(j-1,k) *(ZR(j,k)-ZR(j-1,k))
     2                 -ZB(j,k)   *(ZR(j,k)-ZR(j,k-1))
     3                +ZB(j,k+1) *(ZR(j,k)-ZR(j,k+1)))
   72    CONTINUE
c
         DO 75  k= 2,KN
         DO 75  j= 2,JN
          ZR(j,k)= ZR(j,k)+T*ZU(j,k)
          ZZ(j,k)= ZZ(j,k)+T*ZV(j,k)
   75    CONTINUE
c

c***************************************
c***  KERNEL 21     MATRIX*MATRIX PRODUCT
c***************************************
 1021 DO 21 k= 1,25
      DO 21 i= 1,25
      DO 21 j= 1,n
      PX(i,j)= PX(i,j) +VY(i,k) * CX(k,j)
   21 CONTINUE
```

**Figure 5. Livermore Fortran Kernels**