

**Fermi National Accelerator Laboratory**

**FERMILAB-Conf-92/324**

## **Reducing Communication Inefficiencies for a Flexible Programming Paradigm**

M. Fischler, M. Gao, G. Hockney, M. Isely and M. Uchima

*Fermi National Accelerator Laboratory  
P.O. Box 500, Batavia, Illinois 60510*

November 1992

Presented at *Lattice '92*,  
Amsterdam, Netherlands, September 13-18, 1992



# Reducing Communication Inefficiencies For a Flexible Programming Paradigm

Mark Fischler, M. Gao, G. Hockney, M. Isely, and M. Uchima<sup>a</sup> \*

<sup>a</sup>Fermi National Accelerator Laboratory  
Batavia, IL 60510 USA

The ACPMAPS system at Fermilab has been upgraded to 50 GF by inserting new CPU modules, based on the Intel i860. This ten-fold increase in power, utilizing the identical communications backbone, places the system in a different realm: The transfer latency and overheads are now greater, relative to the cost of a floating point operation. We explore the consequences for programs written using CANOPY, which relies on low communications latencies. We present techniques for alleviating the efficiency decrease, by coalescing transfers, without abandoning the considerable advantages of the CANOPY paradigm.

## 1. ACPMAPS Upgrade and CANOPY

The ACPMAPS [1] project is a collaboration between the Fermilab Computer R&D and Theory departments. The system is designed for both algorithm exploration and lattice gauge calculations. A concept oriented tool set—Canopy [2]—is provided to facilitate natural coding of grid-like problems for the parallel system.

The processing elements of ACPMAPS are 612 nodes each consisting of an Intel i860 CPU, along with 32 Mbytes of DRAM memory. Two independent processor nodes reside on each processor module. The communications scheme is hierarchical in nature—at least 8 processor modules occupy slots in each of 36 crossbar switch crates. The crates are, in turn, interconnected, and automatic routing allows any node to access any other, going through at most one intermediate crate. Thus we have a low latency, “flat global” addressing space, analogous to a phone network.

The Canopy paradigm looks at the problem in terms of tasks done on sites. The sites are organized to form a grid, with some given connectivity (“neighboring sites”). Field data is associated with each site—each site can be considered a “virtual processor” with its own memory for the field data. Logically, all the sites involved in a given task run simultaneously (although of course there are more sites than physical CPU nodes; each

node will handle many sites, one at a time). So for example, during a task there may be occasion to read a field at the site one unit in the X direction away from the site currently being worked on. This data may or may not reside on the local physical node; the Canopy software makes the determination, and either returns a pointer to the requested data, or reads the data from the remote node and returns a pointer to a copy of that data. The user need not be concerned with details of which node owns which sites. Canopy requires MIMD processing and “flat global” access, with a “read/write” communications model—the target node does not know when an access will be made, and need not prepare data in advance.

The physics that has been done on ACPMAPS recently includes B and charmonium physics. Techniques for extraction of measurables have been explored, including operator smearing and higher order terms in fermion propagators [3–5]. Studies done on ACPMAPS have focused on algorithm exploration, and on calculations done with controls on all uncertainties[6]. These studies complement efforts on other special purpose systems, which tend to focus on a few specific calculations. To understand sources of systematic uncertainty, the scientist must explore varied approaches to a given problem. For work of this nature, the convenience and clarity of the Canopy tools were indispensable.

In the 5 Gflop system, communications costs (software overhead required to open channels,

<sup>a</sup>Fermilab is operated by Universities Research Association, Inc. under contract with the U.S. Department of Energy

and the time taken to transmit data) were a minor effect, and contention effects were negligible. We have replaced the CPU modules based on one 20 Mflop processor, with modules containing two 80 Mflop processors. The number of processor modules was increased by 20%; the communications hardware was left unchanged. This upgrade increases computational power by a factor of 10, without improving communication: A detailed analysis of the communications requirements of several algorithms would indicate that communications limitations should now be important, and that contention effects may be critical.

## 2. Efficiency Losses due to Transfer Overhead

Canopy confers many advantages—at a price. It shields the user from the details of the system architecture (while still allowing the scientist to cleanly express the underlying parallelism of the problem). Also, since the number of nodes involved in a job is arbitrary, the system can be shared by multiple users in an automated and efficient manner. The price paid is that for jobs involving a good deal of off-node data access, Canopy applications tend to do quite a few very short transfers, the typical transfer involving field data associated with a single site.

The total internode data transferred for a given algorithm is not impacted by the Canopy paradigm, but the granularity is. Transfer overheads become much more important. Simple overheads can become noticeable fairly quickly, and bottlenecks, saturation and contention issues are of still greater importance. We have investigated these effects for a particularly communications-bound problem (the de-Grand/conjugate gradient method for propagator inversion) and a slightly more CPU-intensive problem (pure gauge Monte-Carlo with Kennedy-Pendleton heat bath algorithm).

Figure 1 shows efficiencies achieved when algorithms are run on increasingly large systems, scaling the lattice size with the number of processors. The lower curve represents the communications-intensive conjugate gradient algorithm, run with no attempt to group transfers. For small num-

bers of nodes ( $< 16$ ), there are fewer off-node accesses required, because the “chunk” of sites owned by each node spans the lattice in one or more directions. By the time 16 nodes is reached, every access across a chunk boundary is off-node, and the number of communications overheads per node per sweep remains fairly constant thereafter. The remaining decrease in performance is due to contention for communications resources.

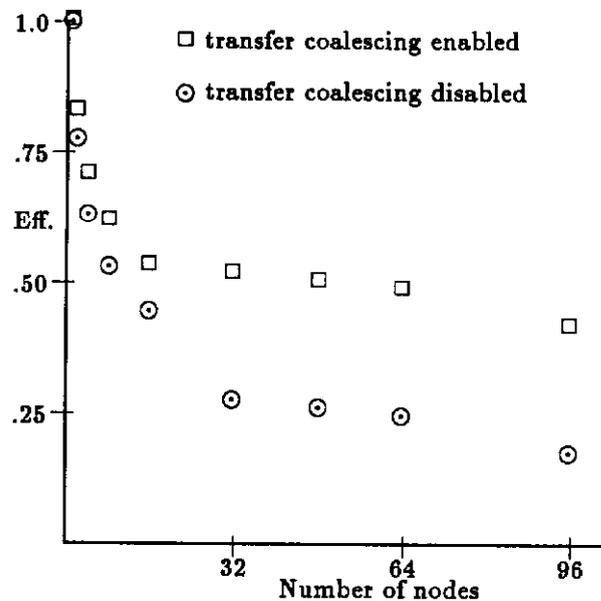


Figure 1. Efficiency vs number of nodes

In ACPMAPS, the key communications resources are the buses connecting crates, since these paths are shared by all the nodes in the connected crates. A naive calculation of bandwidth requirements would indicate only moderate degradations due to communication contention. As shown in figure 1, the contention effects are instead substantial. This is because a fraction of the per-transfer overhead is spent with the bus open—that is, occupying the saturated resource. In fact, for typical 12-18 word transfers, the bus-open overhead is three times as large as the actual data

transfer time. (This is partly due to the nature of the i860 chip—the target “slave” node must participate in the transfer, responding to an interrupt.)

Beyond simple intercrate contention, there can also be worse-than-linear effects for accesses which must traverse multiple crates. When non-linear contention effects are present, the performance can continue to degrade as the problem and system sizes grow larger, in spite of the fact that more communications resources come into play.

### 3. Transfer Coalescing

Both the CPU time lost to communications overheads, and the intercrate bus contention effects, are exacerbated by the fact that the Canopy application tends to do many small transfers. It is beneficial to “coalesce” several accesses to one node into a single data transfer. Several strategies might accomplish this. For example, the user could explicitly guide the communication, having the target nodes prepare large blocks of data for transfer in advance. (Typical applications on message-based systems force the user to do this.) We rejected this approach, since it abandons the Canopy paradigm, and forces the user to substantially change the structure of the program, modifying algorithms to fit the hardware available.

A second approach would be to create bigger “sites” by collapsing one dimension of the problem. For example, a site may now be an entire line of points (X, Y, Z, T) for all values of T. The collapsed problem can still be done using Canopy, and typically will run efficiently since most overheads are amortized over what used to be  $N_t$  sites. However, the user has had to sacrifice part of the convenience of Canopy by altering the program. We have implemented this approach for several production algorithms, to get early estimates as to how much gain can be achieved through transfer coalescing, and to provide efficient versions of early key programs.

The approach we have settled on is “multi-threading.” When a field access routine requires off-node data, the system makes note of the data requested but does not perform the remote access. Instead, sufficient state is saved so that the

processing for this site can continue at a later time, and Canopy moves on to processing another site (“a new thread”). Eventually, remote accesses will be done to satisfy several of the requested accesses at a time. (This requires efficient remote scatter/gather capabilities, which were easy to create for ACPMAPS, and could be implemented for most Canopy platforms.) Once the requested data is present, execution of the suspended threads can continue.

The number of threads which may be active at one time is memory limited. Space must be reserved for the remote data to be gathered, and for stack space for each active thread. For typical applications on ACPMAPS, we use up to 512 threads at one time. The efficiency gain from multi-threading depends on the degree of coalescing—how many short transfers are typically grouped into one longer transfer. (There is a slight per-block cost in the scatter/gather implementation, but this is tiny compared to the communications overhead it replaces).

For any problem limited by communications resource contention, the amortization of resource-open overheads is a pure gain. Once a problem is back in the CPU-limited realm, this gain is diminished by the overhead required to switch contexts from one thread to the next. Fortunately, this does not carry all the baggage normally associated with a context switch: when a task routine has called a field data access routine, only a small amount of state information need be saved to permit suspension of the thread.

For the problems examined, collapsing a dimension and multi-threading each yield comparable efficiency gains—this indicates that the thread-switching overhead is not a severe effect. Since multi-threading involves little user inconvenience, we have incorporated it into Canopy. In most cases the user can get these benefits simply by selecting multi-threading on the command that starts the Canopy job.

There is one subtlety: The Canopy manual warns against using global variables which may be altered inside task routines—there is ambiguity when reading such a variable, as to whether another site has written a new value to it yet. However, global variables which are writ-

ten within a task, used by lower-level subroutines, and discarded when the task moves on to the next site –“task globals”– are logically consistent, but will not work in multi-threaded jobs. Fortunately, in our experience task globals are rarely used, and there is a simple prescription for modifying such variables for use in a multi-threaded context.

#### 4. Results of Multi-threading

The extent to which multi-threading can alleviate communications saturation and overhead effects will depend on what degree of coalescing is achieved. This is application dependent, so we have applied the technique to several types of problems. For fairly local algorithms, the degree of coalescing tends to be around 15% of the maximum number of threads supported. For highly non-local but fairly regular algorithms, such as FFT's, this deteriorates only slightly. Therefore, even if a conservative limit of 100 threads is imposed, the communications overheads are typically amortized over a substantial number of transfer blocks—there is potential for large gains.

These gains are realized in practice. For the highly communications intensive Conjugate-Gradient algorithm, the efficiency of the coalesced version is shown by the upper curve in figure 1. The efficiency levels off once more than 16 nodes are in use. (Even when more than 500 nodes were used, the efficiency remained about 50%). When multi-threading is applied to the pure gauge Monte-Carlo, efficiencies level at 63% for large numbers of nodes.

These tests studied how efficiencies scale with problem and system size. The context switching mechanism used was non-optimized, and took CPU time approximately equal to the communications overhead replaced. Thus the intercrate contention effects are isolated, primarily affecting the behavior when more than 16 nodes are used. Once thread context switching is optimized, the efficiency drop between 1 and 16 nodes will be much smaller, and efficiencies for large numbers of nodes will improve by at least 15 - 25%.

With multi-threading in place, the drop-off in efficiency when progressing from a crate of nodes to the entire system is not large. This indicates

that non-linear contention effects due to multi-hop communications are not severely impacting performance. Without multi-threading, these effects become disastrous as the number of nodes increases. The ACPMAPS system does automatic spooling and assigning of node resources. Since the efficiency is a fairly flat function of number of nodes, it becomes possible to do a good job of resource allocation.

Several systems other than ACPMAPS are plausible Canopy platforms, but were not designed with low-latency communications in mind. For example, the Intel DELTA has a latency for read accesses of about twice that of ACPMAPS, because two separate messages must be handled. Transfer coalescing techniques should allow users of such systems to get the convenience and benefits of the Canopy tool set, while retaining reasonable efficiency.

#### 5. Conclusions and Acknowledgments

The point of the upgraded ACPMAPS system is to do lattice gauge physics, not to study computer science issues. For some applications, the impact of internode communications is not severe. But for the most effective use of a system of this nature, efficient running of tightly-coupled applications must be supported. We have presented a software technique in this area.

The authors would like to acknowledge the contributions of the lattice gauge physicists using the system [3, 4, 6], who supplied codes on which to test the effects of these techniques, and who have had input into the strategies chosen.

#### REFERENCES

- 1 M. Fischler, FERMILAB-TM-1780 (1992)
- 2 Canopy 5.0 Manual, M. Fischler, G. Hockney, P. Mackenzie, available from the Fermilab Computing Division
- 3 A. El-Khadra, these proceedings
- 4 E. Eichten, these proceedings
- 5 H. Thacker, these proceedings
- 6 A. El-Khadra, G. Hockney, A. Kronfeld, P. Mackenzie, Phys. Rev. Letters 69,729 (1992)