Sparse Networks: balance of processing and communication

Ville Leppänen and Martti Penttonen

Department of Computer Science
University of Turku
Lemminkäisenkatu 14a, 20520 Turku, Finland
Department of Computer Science,
University of Kuopio
P.O.Box 1627, 70211 Kuopio, Finland

E-mail: Ville.Leppanen@utu.fi, penttone@cs.uku.fi

Abstract

We discuss the problematics of efficient, general purpose parallel computation. Parallel processing is meaningful only if latencies in data access and interprocessor communication are managed. Latencies can be compensated by slackness. This requires highly parallel algorithms, and sufficient communication bandwidth. We present a simple criterion for the need of bandwidth, or reversely, for the amount of processors a network can serve. This leads to the concept of sparse network. We show that sparse tori provide a basis for efficient parallel computation.

1 Introduction

In this article we study the need of communication in parallel computing and investigate methods to achieve a balance between processing and communication so that computation is fast and the use of hardware resources is efficient. We point out the importance of sparseness of networks in this game.

Until recent years the performance of processors has exponentially grown by shortening the clock cycle and using more sophisticated processor architectures. Also the packing density of the memory has grown [4].

Quite recently, however, something has changed. Shortening the clock cycle and increasing the packing density is no more as cost effective if possible, and heatening has become a problem. In this situation, processor industry has started to look for other paths. Additional performance is not searched by speeding up a single processor but by multiplying processing units. Multicore processor is the keyword of this development.

Multiple processors alone do not multiply processing power. Processors must be able to cooperate and the communication of the processors must be fluent enough. Even more important, we must be able to program our problems in such a way that programs can be automatically transformed to efficient execution by processors. This includes the following challenges:

1. algorithmic difficulty
2. compilation of programs for efficient execution in network
3. performance of the network connecting the processors and memories
4. balance of processor load

Some of these challenges have at least a theoretical solution, some others are less obvious. There is a developed theory of parallel algorithmics, even if it is not very well known to the main stream computer scientists [3]. The challenge of automatic compilation and efficient execution are much harder than in the case of sequential computation, because there are so many things to take into account at the same time: multiple processors, properties of the network etc. In lack of general solution, these nasty problems are left as the responsibility of the poor programmer. The focus of this article is the balance of network capacity and the processing power. We assume that the balance between the processors is solved at higher levels, algorithmics and compilation.

2 Latency and slackness

In general, a good strategy to finish a job faster is to add work force. In computer, processors are the work force. However, more work force means more administration and communication. Unless computation cannot be

split to completely independent subcomputations, processors need to send data to each others, see Fig. 2. A computation cannot continue, before the necessary data are available for the processor. This is called *latency*.

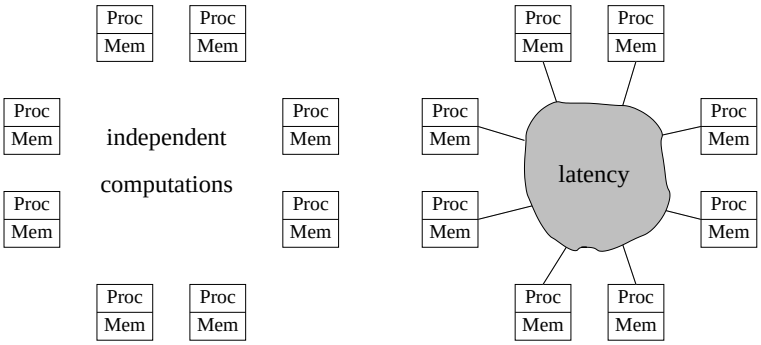

Figure 1: (a) sequential computations (b) parallel computation

Latencies in networks are unavoidable. When the number of processors grows, the network grows, and the latency grows. Fortunately, latency can be compensated, assuming that there is enough parallelism and enough bandwidth available.

Assume that a parallel algorithm can efficiently use sp "virtual" processors, where p is the number of "physical" processors and s is a multiplier called *slackness* [5]. Share the virtual processors to physical processors so that each physical processor gets s virtual processors (or *processes*) it is responsible for. The physical processor runs the s processes in turn, so that there is an interval of s time units between two successive steps of the virtual processor. This time can be used for communication. Thus slackness can be used to cover the latency.

3 Need of bandwidth

The slackness works assuming the communication network allows to retrieve the required by s processes in $O(s)$ units of time. As all processors

are running at the same time, there is a high need of bandwidth in the network.

Let us take a closer look at the situation in Fig. 2 (b). Assume that the computation can exploit all processors but we cannot assume locality. Hence, any processor may need to communicate with any other processor at any moment. In real world we must accept that it needs some time and physical resource to get data from another processor. In 3-dimensional world the distances of the processors grow with factor $\Omega(\sqrt[3]{p})$, where p is the number of processors [6]. When clock cycle is of the order of 1 GHz, physical distances matter. Also it is not realistic to assume that unbounded number p of processors be pairwise connected. Some intermediate nodes are needed. Let ϕ denote the time needed by the data access. It may be the maximum number of node-to-node hops in the network (i.e. the *diameter* of the network) or some other network dependent delay. Furthermore, assume that every k'th step of computation needs communication with another processor. Under these assumptions, we get the condition

$$p\phi/k \leq C \tag{1}$$

for the necessary total *network capacity* C. C depends on the topology and the technology of the network; typically it is something like the number of connecting wires in the network. If (1) is not fulfilled, slackness does not help to cover the latency.

Network capacity condition (1) leads us to the discussion, what kind of network is required, or how many processors a network can serve.

4 Dense and sparse networks

We call a processor network *sparse* (or sparsely populated) if only a fraction of nodes are *processors*. Nodes that are not processors, are called *routers*. An example of dense and sparse network is found in Fig. 4.

Sparse networks are interesting because of the bandwidth capacity condition (1). Processors are not useful unless the network can transport data needed by the computation.

We will take a closer look at meshes and tori. Meshes are interesting networks, because they have a simple, regular, scalable structure, and they can be built in 3-dimensional world with wires of constant length.

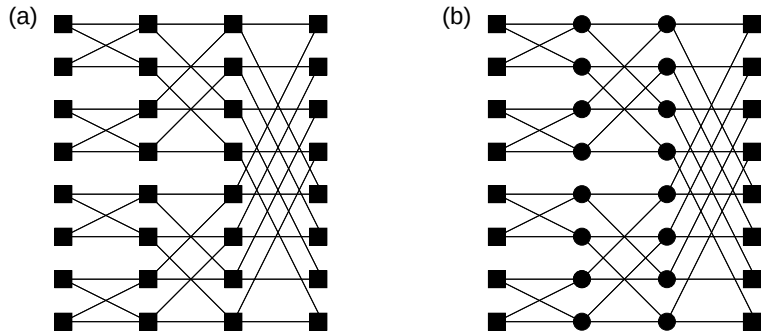

Figure 2: (a) Dense butterfly (b) Sparse butterfly. Squares are processors and disks are routers.

The nodes of a d-dimensional *(dense) mesh* $M(n,d)$ are d-tuples $(x_1,\ldots,x_d)$, where $0 \le x_i < n$. A node $(x_1,\ldots,x_i,\ldots,x_d)$ is connected to $(x_1,\ldots,x_i+1,\ldots,x_d)$, if $x_i < n-1$. A *torus* $T(n,d)$ is like mesh, but in addition $(x_1,\ldots,n-1,\ldots,x_d)$ is connected with $(x_1,\ldots,0,\ldots,x_d)$.

In a *sparse* mesh (or torus), only nodes fulfilling

$$x_1 + \ldots + x_d \equiv 0 \bmod n$$

are processors, other nodes are routers. For examples, see Figure 4

Let us apply the bandwidth condition (1) to the s-sided mesh $M(s,2)$, where all s^2 nodes are processors, i.e. $p = s^2$. The diameter of the network is $\phi = 2s - 2 \approx 2s$. If every step of computation needs communication, then $k = 1$. Hence the left hand side of (1) is about $2s^3$. The out-degree of every node is 2, thus $C = 2s^2$. Hence condition (1) is not fulfilled. There is not enough bandwidth for so many processors.

Now, consider the case of the sparse mesh $SM(s,2)$, where $\phi = s$, $p = s$, and the number of nodes is s^2. In this case, for $k = 1$ the left hand side of 1 is about s^2, while the right hand side remains the same $2s^2$ as in dense mesh. Thus (1) holds.

This pair of examples shows us that if processors are packed densely, the need of communication may be too high for the bandwidth of the network. It is better to replace a large part of processors by routers and

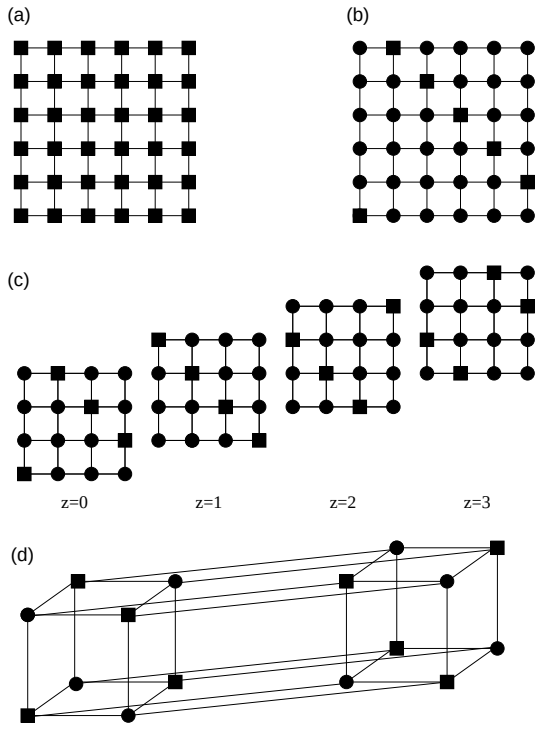

Figure 3: Mesh (a) $M(6,2)$, and sparse meshes (b) $SM(6,2)$, (c) $SM(4,3)$, (d) $SM(2,4)$

in this way balance the processing power and communication bandwidth. Fortunately this kind routers are very simple components in comparison with processors.

If an application is not extremely intensive in communication, increasing k may help to fulfil (1). Also if communication can be made more local, ϕ gets smaller and even a dense network may work.

5 Running parallel programs on $ST(n, d)$

Bandwidth condition (1) is a necessary condition for fast enough communication, it is not a guarantee. For each network we must verify that that network can route s messages per processor in $O(s)$ time. In this section we shall show that it is possible to efficiently route messages and run parallel programs in $ST(n, d)$.

First, consider $ST(n, 2)$, where routers are as in Fig. 5 [1]. The routing is very simple, indeed, by the following rules:

1. Routers are in the crossing state at time moments 0, n, $2n$, ..., and in the direct state else
2. At moment t, processor $(x, n-x)$ sends a packet along x-axis to the processor $(x + n - t, t - x)$ if it has one to be sent

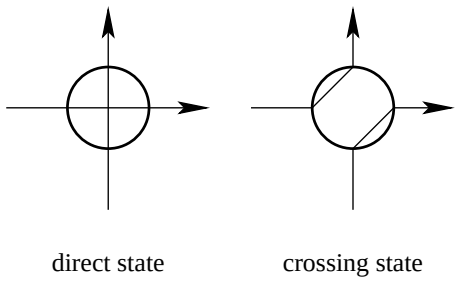

Figure 4: Two states of the router

The point of this algorithm is that packets do not collide and thus each sending is successful. Also, at every moment, another packet can be sent along the y-axis. If randomized hashing is used for memory mapping, packets are sent evenly, and with high probability, in time $n \log n$ each processor will generate $\Theta(\log n)$ packets to be sent for each of the processors. They can be routed in $\Theta(\log n)$ time by the above described routing algorithm.

Routing in $ST(n, 2)$ generalizes to higher dimensions. One may ask, if $ST(n, d)$ for $d > 3$ is useful in 3-dimensional world. In theory, the advantage of higher d is that the number of processors grows with n^{d-1} while

diameter of the network grows with $n(d-1)$. For example, $ST(n,3)$ has n^2 processors and interprocessor distances are n or $2n$. A new difficulty in comparison with $ST(n,2)$ is that interprocessor distance is no more constant.

For example, consider $ST(3,4)$. The processor-to-processor distance is 0, 3, or 6. Processor-to-processor paths can be described with *path patterns*, which are d-tuples telling how big is the shift in each dimension. In $ST(3,4)$ the path patterns of all processor-to-processor paths are.

```
0000,
0012, 2001, 1200  0120
0021, 1002, 2100, 0210,
0102, 2010, 0201, 1020,
0111, 1011, 1101, 1110,
0222, 2022, 2202, 2220,
1212, 2121,
1221, 1122, 2112, 2211,
```

In this case, there is one pattern (i.e. packet) for distance 0, 16 patterns for distance 3 and 10 patterns for distance 6, altogethern $27 = 3^3$ patterns, so all processor-to-processor paths are listed.

In our example, we grouped the pattern paths in a special way: Patterns that are permutations of each others are grouped together (on one line). This leads to the following grouping strategy. Consider the pattern group 1221, 1122, 2112, 2211 for example. If we start sending 1221 along dimension 1, 1122 along dimension 2, 2112 along dimension 3 and 2211 along dimension 4, packets turn at the same time. Due to permutation property, on all paths in a group, the turnings occur at the same time, to the cyclically higher dimension. Therefore these packets do not collide. Hence, we can focus on routing one representative in each group.

By previous observation, the routing task reduces to routing 0000, 0012, 0021, 0102, 0111, 0222, 1212, 1221. Obviously, we can forget 0000, because it is a packet to the processor itself. By the greedy principle, farthest going packets first, we come to the schedule of Figure 5. In this case, all 27 packets could be sent in 12 time units.

It was proved in [2] that the above described routing algorithm routes packets in time less than $p/2$ assuming each processor has p packets ad-

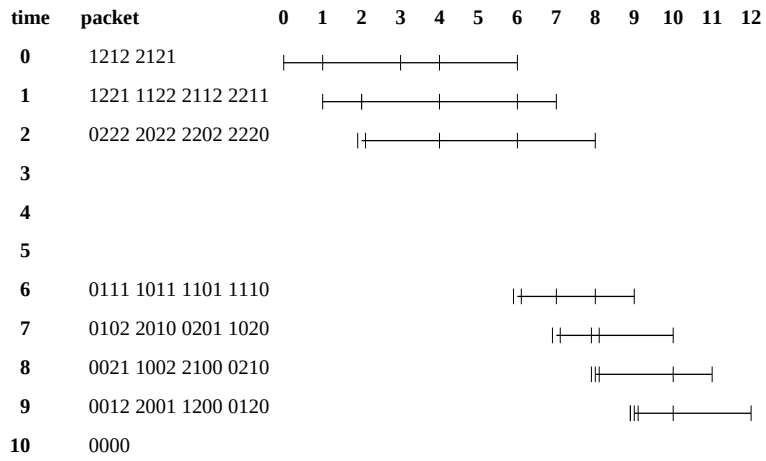

Figure 5: Schedule for $ST(3,4)$. In each group we show the first packet. Note that at moments 3, 4, and 5 a new packet cannot be sent because previous packets are still using the link. By picture, all 27 packets have arrived at targets by time moment 12. Hence, the time cost per packet is $12/27$.

dressed to different processors. Hence, if processors have $\Omega(p \log p)$ packets, with high probability routing cost per packet is $1/2$.

6 Conclusions

In this survey we demonstrated that time demanding computational problems can be resource efficiently computed if there is a parallel algorithm providing sufficient parallel slackness. However, slackness only helps, if the network provides enough bandwidth. The processing power and bandwidth can be balanced by using sparse networks. We saw that sparse meshes and sparse tori are suitable topologies for parallel computers. The concepts of latency - slackness - bandwidth - sparseness are universal and apply to different technologies. With them it is possible to balance the processing power with communication, which is the key for resource efficient computation.

References

[1] R. Honkanen. Nearly-All-Optical Routing in Sparse Optical Tori. In *Proc. 5th Int. Conf. on Parallel Computing in Electrical Engineering*, PARELEC 2006: 251-256, IEEE Computer Society, 2006.

[2] R. Honkanen, V. Leppänen, M. Penttonen. Address-free all-to-all routing in sparse torus. In *Proc. Int. Conf. on Parallel Computing Technologies, PACT'2007*, LNCS 4671:200-205.

[3] J. Jájá. em An Introduction to Parallel Algorithms. Addison-Wesley, 1992.

[4] G.E. Moore. Cramming more components onto integrated circuits *Electronics 38(8)*, 1965.

[5] L.G. Valiant. General Purpose Parallel Architectures. In *Algorithms and Complexity, Handbook of Theoretical Computer Science*, volume A, pages 943–971, 1990.

[6] P.M.B. Vitányi. Locality, Communication, and Interconnect Length in Multicomputers. *SIAM Journal on Computing*, 17(4):659 – 672, August 1988.