Maciej Drozdowski

Scheduling for Parallel Processing

Springer

the star topology. This procedure applied recursively unfolds the tree and determines the load distribution for a whole tree. This method has been applied e.g. in [9, 11, 12, 14, 31, 112].

Divisible load processing with or without communication front end, with or without returning of the results, has been analyzed in [35]. Low-order complexity heuristics to find the set of active processors and communication sequence were proposed in [6]. Specific trees were foundation of load scattering in other interconnection topologies. We will present them in the following text.

7.2.3 Meshes

In this section, we assume that the meshes are homogeneous. The computing rate is A, and communication link parameters are C, S, for communication rate and start-up time, respectively. Processors can communicate and compute simultaneously. Originator P_0 takes part in the computation. The time of returning the results is negligible.

Divisible load processing algorithms for meshes were built on the basis of various load scattering algorithms. The first publication on divisible load scheduling in meshes [28] assumed that the originator is located in the center of a 2D-mesh, communication start-up costs is negligible ($S = 0$), the load is distributed in a store-and-forward manner from a processor to a processor. The sets of processors activated in the same iteration of load distribution formed *layers* of rectangular shape around the originator (Fig. 7.4a). The whole layers may be treated as processors connected in a chain. By the use of the equivalent speed of the whole layer of processors, performance limits of a 2D-mesh network with store-and-forward load scattering have been analyzed (compare the analysis for a chain in Sect. 7.12.2).

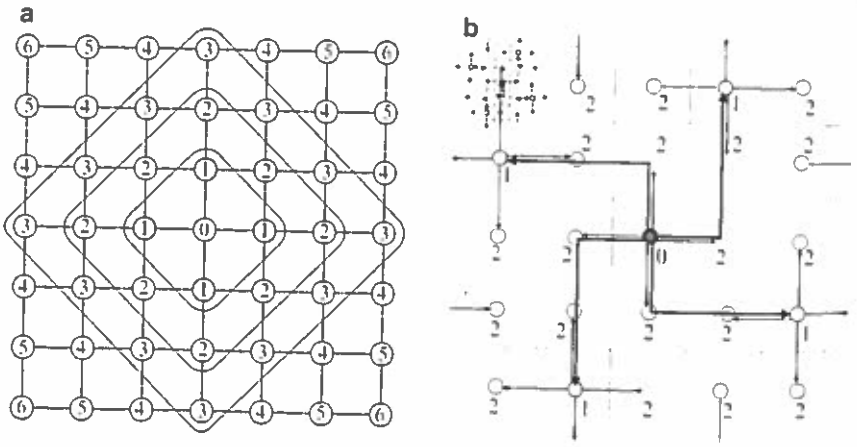

Fig. 7.4 Load scattering in a 2D-mesh (a) using store-and-forward routing and (b) using circuit switching and $p = 4$ ports

A different set of scattering algorithms were proposed for circuit-switched and wormhole routing. In circuit switching and wormhole routing the communication delay does not depend on the distance between the sender and the receiver (cf. Sect. 3.1.3). As a result it is possible to send the load very far from the originator, and then to redistribute it in remote parts of the network. A 2D-mesh broadcasting algorithm taking advantage of such a structure of communication delay was proposed in [91]. It was adopted as a scattering algorithm for divisible loads, and generalized in a series of publications [3, 39, 40, 52, 58]. The method of load scattering for 2D-toroidal mesh, based on [91] and used in [3], is presented in Fig. 7.4b. Each scattering iteration has two steps: chess queen moves and then cross moves using the torus wrap-around connections. Note that the pattern of load distribution recursively repeats itself in submeshes of five times smaller side size. Furthermore, all communication ports of the already activated processors are used in each step of load distribution. Let p denote the number of processor ports used in each step of load distribution. Examples of load scattering patterns for different numbers of ports p, 2D-, and 3D-meshes are shown in Fig. 7.5. Also here the communication path patterns are repeated in submeshes with $p + 1$ times shorter side size, and all p ports of the active processors are busy in each communication step. More examples for 3D-meshes and various number of ports p used simultaneously can be found in [39].

Let us now make some observations on the general structure of the scattering patterns presented in Figs. 7.4b and 7.5. The processors act synchronously, and all processors activated in iteration i receive load simultaneously. In the first scattering step the originator distributes load to p processors. In the next step each activated processor sends load to p new processors. Thus, $p(p + 1)^{i-1}$ processors receive load in step i of scattering. At the end of step i the number of active, and hence

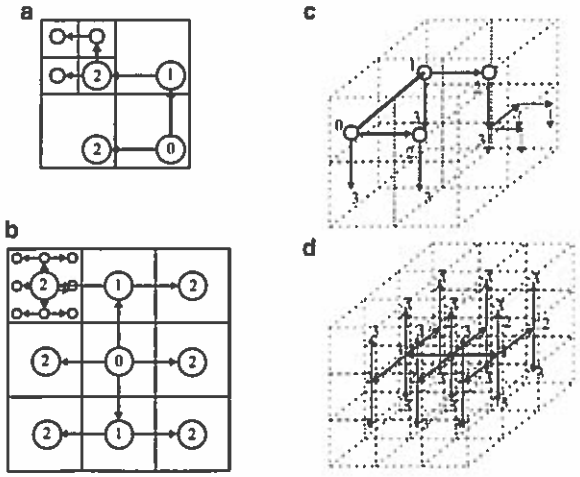

Fig. 7.5 Load scattering in meshes using circuit switching, and different numbers of ports p: (**a**) 2D-mesh, $p = 1$ (**b**) 2D-mesh, $p = 2$ (**c**) 3D-mesh, $p = 1$ and (**d**) 3D-mesh $p = 2$

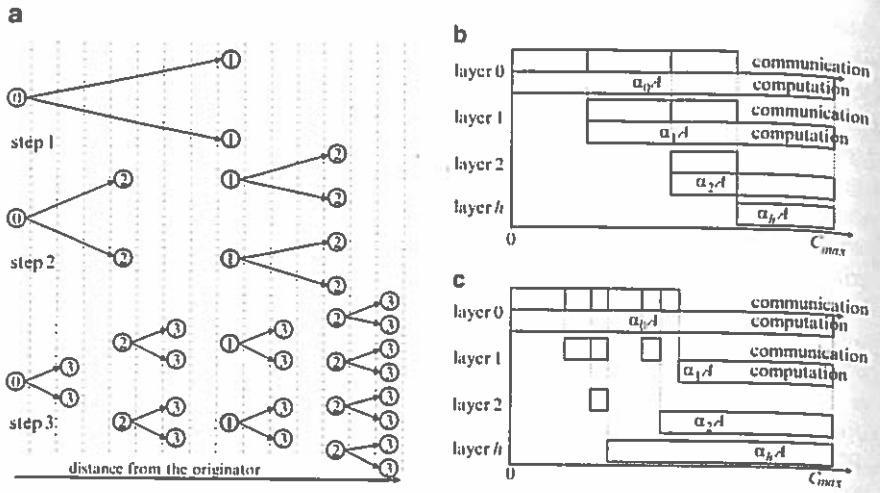

Fig. 7.6 (a) 3-Nomial tree spread along the distance in the network. (b) A schedule for simple load scattering in $(p+1)$-nomial tree. (c) A schedule for LLF scattering strategy

already computing, processors is $(p+1)^i$. The scattering can be viewed as a tree which can be called a $(p+1)$-nomial tree. Let us call a *layer* the set of processors activated in the same step, and hence on the same level of the $(p+1)$-nomial tree. Let $h = \log_{p+1} m$ be the last level in the tree (the tree has $h+1$ levels including the root at level 0). Each node at levels $0, \ldots, i < h$ has exactly p successors at level $i+1$. This is illustrated in Fig. 7.6a. A processor activated in iteration $i < h$ has successors in layers $i+1, \ldots, h$. Each of the successors in layers $i+1, \ldots, h$ must receive from a node in layer i load not only for itself, but also for its successors. Hence, a processor receives load once, but must redistribute it in many steps. Let α_i denote the load computed by each processor in layer i. Then, a processor in layer i receives load of size $\alpha_i + p\sum_{j=i+1}^{h}(p+1)^{j-i-1}\alpha_j$. Having received the whole message with the load a processor in layer i starts computing its part α_i, and redistributing load $\alpha_{i+1} + \sum_{j=i+2}^{h}(p+1)^{j-i-2}\alpha_j$ via each of its p ports. It can be observed that layer i computes as long as sending load to layer $i+1$, and computing on layer $i+1$ (cf. Fig. 7.6b). Hence, load distribution can be calculated from the following linear system:

$$\alpha_i A = S + C(\alpha_{i+1} + p\sum_{j=i+2}^{h}(p+1)^{j-i-2}\alpha_j) + \alpha_{i+1}A, \quad \text{for } i = 0, \ldots, h-1 \tag{7.25}$$

$$V = \alpha_0 + p\sum_{i=1}^{h}(p+1)^{i-1}\alpha_i \tag{7.26}$$

Closed-form solutions for the above equations were derived in [40].

A different load distribution strategy on $(p+1)$-nomial trees has been proposed in [58] and later in [52]. Instead of starting computations as soon as any load arrives to a processor, the whole communication network is exploited to activate selected layer of processors. It has been shown [58] that *Largest Layer First* (LLF) method activating the layers in the order of decreasing number of processors (see Fig. 7.6c) has the shortest schedule length among all possible activation orders. Let us analyze communication time in activating layer i. Layer i receives load in i steps. Processors in layer i receive load $p(p+1)^{i-1}\alpha_i$ in total. In the first step originator sends load $p(p+1)^{i-2}\alpha_i$ over each of its p ports to layer 1 which will be redistributed from layer 1, via layers $2,\ldots,i-1$ to layer i. The remaining load $p(p+1)^{i-2}\alpha_i$ is sent from the originator to layer i via layers $2,\ldots,i-1$, bypassing layers $1,\ldots,i-2$, respectively. More generally, in step $j < i$ all active processors send to layer j load of size $p(p+1)^{i-j-1}\alpha_i$. Hence, the total time of communication to layer i is $Si + C\alpha_i(1 + p\sum_{j=1}^{i-1}(p+1)^{i-j-1}) = Si + C(p+1)^{i-1}\alpha_i$. The load partitioning between the processors can be found by solving the following set of equations:

$$\alpha_0 A = Sh + C(p+1)^{h-1}\alpha_h + \alpha_h A \tag{7.27}$$

$$\alpha_i A = S(i-1) + C(p+1)^{i-2}\alpha_{i-1} + \alpha_{i-1}A, \quad \text{for } i = h,\ldots,2 \tag{7.28}$$

$$V = \alpha_0 + p\sum_{i=1}^{h}(p+1)^{i-1}\alpha_i \tag{7.29}$$

It was shown in [52] that if messages have restricted sizes imposed, e.g., by limited sizes of communication buffers, then LLF is not always the best load scattering strategy. In this case *Nearest Layer First* (NLF) activation order which is opposite of LLF may be advantageous.

7.2.4 Hypercubes

In this section, we assume that the system is homogeneous, processors can communicate and commute simultaneously, originator takes part in the computation, the time of returning the results is negligible. Similarly to the networks presented above, a load scattering algorithm is necessary as an intermediary between the interconnection topology and divisible load processing model. The load in a hypercube may be distributed by activating processors along consecutive hypercube dimensions. For example, P_0 sends load to processor P_1 along dimension 0, then P_0, P_1 send load to P_2, P_3, respectively, along dimension 1, etc. This scattering method uses binomial tree embedded in the $\log m$-dimensional mesh of side size 2. Note that this is a special case of $(p+1)$-nomial tree, for $p = 1$, presented in Sect. 7.2.3. A disadvantage of this approach is that only one path is used to activate a processor.

Hypercube scattering algorithms taking advantage of dense network structure have been proposed in [27, 39]. We will present here only two simple algorithms. Before proceeding with further presentation let us remind some features of hypercube network (cf. Sect. 3.1.3). Let $d = \log m$. Each processor of the hypercube may

be labeled with d-bit binary number such that neighboring nodes differ in exactly one bit. Thus, there are $\log m$ ports in each node. Suppose originator P_0 has label 0. Then, its direct neighbors have exactly one bit equal 1 in their labels. The processors in a distance of two hops from the originator have two bits equal to 1 in their labels. Analogously, processors with i 1s in their labels are i hops away from the originator. Let us call a *layer* the set of processors in equal distance from the originator. The number of processors in layer i is $\binom{d}{i}$. Since the neighboring processors differ only in one bit, each node in layer i can be reached via i ports. A processor in layer i may send load to layer $i+1$ through $d-i$ ports.

In the first scattering algorithm load is passed from layer to layer in a store-and-forward manner as if in a chain. The originator sends the load for layers $2,\ldots,d$ over its d ports in the first communication step. Then processors in layer 1 (counting from 0) intercept their load part α_1, and relay the remaining load to layer 2. In the same way processors in layer $i > 1$ capture α_i units of the load from the received messages, and send the remaining load to layer $i+1 \leq d$. When held in layer i the load for layers $i+1,\ldots,d$ is equally distributed between all $\binom{d}{i}$ processors of layer i. This load is relayed in equal sizes to $d-i$ ports. Since the time of returning the results is negligible, all processors should stop computing simultaneously. Consequently, the time of computing in layer i is equal to the time of sending to layer $i+1$, and computing on it. Load partitioning can be calculated from the following linear system:

$$\alpha_i A = S + C\frac{\sum_{j=i+1}^{d}\binom{d}{j}\alpha_j}{(d-i)\binom{d}{i}} + \alpha_{i+1}A \quad \text{for } i = 0,\ldots,d-1 \tag{7.30}$$

$$V = \sum_{i=0}^{d}\binom{d}{i}\alpha_i \tag{7.31}$$

A different scattering algorithm takes advantage of circuit-switching routing and partitions the load in the intermediate layers without storing it. A processor in layer i, having received its part of the load from i predecessors, immediately starts computing. The remaining load is relayed to $d-i$ output ports in equal shares without storing in layer i. Hence, the messages to layers $i+1,\ldots,d$ reach their destinations nearly immediately after departing the originator, and the bandwidth of the d output ports of the originator is determining the communication delay. Hence, the distribution of the load can be found from the following equations:

$$\alpha_i A = S + \alpha_{i+1}\left(\frac{C}{d}\binom{d}{i+1} + A\right) \quad \text{for } i = 0,\ldots,d-1 \tag{7.32}$$

$$V = \sum_{i=0}^{d}\binom{d}{i}\alpha_i \tag{7.33}$$

Other methods of divisible load scheduling in hypercube networks were considered in [39].