Three non Conventional Paradigms of Parallel Computation*

Fabrizio Luccio[1], Linda Pagli[1] and Geppino Pucci[2]

[1] Dipartimento di Informatica, Università di Pisa, Pisa, Italy
[2] Dipartimento di Elettronica e Informatica, Università di Padova, Padova, Italy

Abstract. We consider three paradigms of computation where the benefits of a parallel solution are greater than usual. Paradigm 1 works on a time-varying input data set, whose size increases with time. In paradigm 2 the data set is fixed, but the processors may fail at any time with a given constant probability. In paradigm 3, the execution of a single operation may require more than one processor, for security or reliability reasons. We discuss the organization of PRAM algorithms for these paradigms, and prove new bounds on parallel speed-up.

1 Introduction

The theory of parallel algorithms has a well known body, developed on the PRAM model [5]. Some folklore principles are at the base of this theory, in particular the ones that express upper and lower bounds on the processing time. Let Π be a problem of size N, and let $T^s(N)$ be the time required by the best known sequential algorithm A^s to solve Π. Any parallel algorithm A^p that solves Π with a number P of PRAM processors requires time $T^p(P,N)$ such that:

$$T^p(P,N) \geq \frac{T^s(N)}{P}. \tag{1}$$

Relation (1) expresses a renowned lower bound on parallel speed-up. The quantity $W(P,N) = T^p(P,N) \cdot P$ is called the *work* of A^p. If the operations of a parallel algorithm with P' processors are rescheduled into another algorithm with $P < P'$ processors, we have:

$$T^p(P,N) \leq \frac{W(P',N)}{P} + T^p(P',N), \tag{2}$$

that expresses Brent's principle on scaling.

Non conventional studies on parallel speed-up have been reported for example in [2], [11] and [7]. In particular, it has been shown that the above relations do not hold for specific classes of problems, if expressed in absolute terms [2], or in asymptotic terms [7]. This paper is aimed to provide a critical contribution to this area. In particular, we consider three algorithmic paradigms where the power

* This work has been supported by MURST of Italy under a research grant.

of parallel computation is exploited to the extent that inequalities (1) and (2) may be violated, and discuss the organization and analysis of algorithms for these paradigms. We have:
Paradigm 1. The input data set varies dynamically. The problem size N is defined as a non decreasing function of time. Intuitively an algorithm performs better than usual for increasing P, because the the processing time decreases and the problem size to be considered is smaller.
Paradigm 2. The processors can fail with constant probability. Again a parallel computation may become much faster with increasing P, because the number of non faulty processors decreases with time.
Paradigm 3. A single operation may require $k > 1$ processors to be executed. This situation may occur for security or reliability reasons. Trivially, the problem cannot be solved with less than k processors.

2 Computing with time-increasing data (Paradigm 1)

Let Π be a problem of size $N = n + \mathrm{f}(n, t)$, where $\mathrm{f}(n, t)$ is a non decreasing function of the initial size n and time t, $\mathrm{f}(n, 0) = 0$. An algorithm A for Π terminates when all the data currently arrived have been treated, that is, the paradigm applies when a condition of consistency is required on a current set of data before an answer can be supplied.

The variability of N occurs for *on-line* problems, where it is assumed that an action has to be taken for each new datum before another datum arrives; and for *real-time* problems, where the time required by any of the above actions is upper bounded by a constant [9]. We study the impact of the variability of N on the design and complexity of algorithms, without imposing the above limitations. New data can be accumulated without being treated immediately, and the time needed to treat them is an arbitrary function of t. Moreover, our approach does not fall in the theory of queueing systems because the data arrival times are known deterministically and no fixed queueing discipline is imposed for their future use.

Many problems in Applied Physics work on variable data which obey to relations similar to the one given above for N, often in differential or integral form. For example the vacuum is made in a chamber while an imperfect valve lets a certain amount of air to enter again. The process terminates when a given condition is met (e.g., the pressure reaches a given value); at this point the valve is tapped and a different experiment can be initiated. In the theory of computing, interesting problems in Paradigm 1 are the ones that admit a sequential *data-accumulative* algorithm A^s (shortly *d-algorithm*). A^s is built for time-increasing data, but its time complexity $T^s(N)$ is the same, in order of magnitude, of the best algorithm working on the N data as if they were all available at time $t = 0$. One such problem admits an optimal parallel solution with P processors if there is a parallel algorithm A^p (*pd-algorithm*) working on time-increasing data, whose complexity is $T^p(P, N) = O\left(\frac{T^s(N)}{P}\right)$. For simplicity, we restrict our study to problems of the class $\mathcal{P}$ admitting a d-algorithm and a pd-algorithm, such

that, for c and c_p constant:

$$N = n + knt, \tag{3}$$

$$t = T^s(N) = c(n + knt)^\alpha, \alpha \geq 1, \tag{4}$$

$$t_p = T^p(P, N) = \frac{c_p(n + knt_p)^\alpha}{P}, \text{ with } P \text{ constant or } P = n^\delta, \delta \leq \alpha. \tag{5}$$

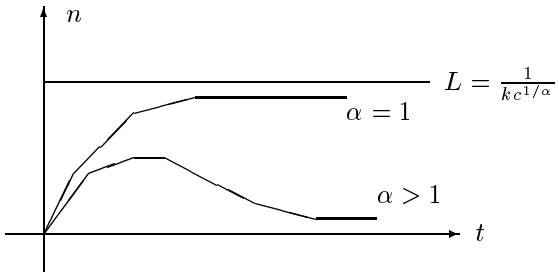

Figure 1

Figure 1 shows the plot of (4). For $\alpha = 1$ the algorithm terminates iff $n < \frac{1}{kc^{1/\alpha}}$, while the working time may become arbitrarily large for n approaching $\frac{1}{kc^{1/\alpha}}$. For $a > 1$ the behaviour is less favourable, because the amount of new data increases rapidly with t. For $P =$ constant, function (5) has a plot in n, t_p like the one of figure 1, with $L = \frac{(Pc/c_p)^{1/\alpha}}{kc^{1/\alpha}}$. For $P = n^\delta, \delta < \alpha$, the plot is the same, with $L = \frac{p^{1/(\alpha-\delta)}}{{c_p}^{1/(\alpha-\delta)}k^{\alpha/(\alpha-\delta)}}$. For $P = n^\alpha$, relation (5) becomes $t_p = c_p(1 + kt_p)^\alpha$, that is t_p is constant for proper values of c_p, k and α (this is consistent with the fact that the parallel algorithm is optimal and the sequential algorithm has complexity n^α). Note that relations (3) and (4) have been formulated under the hypothesis that the new knt data are loaded into memory without increasing the time complexity of the algorithm. It can be easily shown that this can be attained for $\alpha \geq 1$ as posed in (4).

The lower bound (1) on parallel speed-up does not apply to the problems solvable in Paradigm 1, where the benefits of a parallel solution may be stronger and more intrigued than usual. In general we can prove that:

Proposition 1. *For problems obeying relation (3), (4) and (5) we have* $\frac{T^s(N)}{P \cdot T^p(P,N)} > \frac{c}{c_p}$. *Furthermore, for* $\alpha = 1$ *the ratio* $\frac{T^s(N)}{P \cdot T^p(P,N)}$ *tends to* ∞ *for* n *tending to* $\frac{1}{ck}$.

Proposition 1 violates relation (1) for $c = c_p$, or for $\alpha = 1$ and n close to $\frac{1}{ck}$. For complexity functions different from (5), Proposition 1 must be reformulated (e.g., $P = \frac{n}{\log n}$ and $t_p = c_p \log n$, see below).

Another bound that does not apply in the framework of Paradigm 1 is the one expressed in Brent's principle (2). For brevity we restrict our discussion to the subfamily of pd-algorithms obeying relation (5) with $\alpha = 1$. Let A' be the fastest known such algorithm with P' processors. A' requires time

$$t'_p = \frac{c'_p(n + knt'_p)}{P'}. \tag{6}$$

For $P < P'$ consider a new algorithm A obtained from A' by rescheduling the operations of A'. Algorithm A requires time $t_p \geq t'_p$, hence $N = n + knt_p \geq n + knt'_p = N'$. However, to build A on N data, we have to reschedule A' on the same amount N of data, which in turn requires that A' be executed for a proper initial value n'' such that $N'' = n'' + kn''t''_p = N = n + knt_p$, with $t''_p = \frac{c'_p(n''+kn''t''_p)}{P'}$. We have:

$$t''_p = \frac{c'_p(n + knt_p)}{P'}. \tag{7}$$

Let X'' denote the total number of operations performed by A' on N'', and let $W''(P', N'') = t''_p \cdot P'$ be the corresponding work. Algorithm A is obtained by rescheduling the X'' operations of A'. As in the derivation of Brent's principle we obtain:

$$t_p \leq \frac{X''}{P} + t''_p \leq \frac{W''(P', N'')}{P} + t''_p = \pi t''_p, \text{ with } \pi = \frac{P' + P}{P}. \tag{8}$$

From relations (7) and (8), we easily derive a new formulation of Brent's principle:

Proposition 2. *A pd-algorithm obeying relation (5) with $\alpha = 1$, constant c'_p, and P' processors, can be rescheduled for $P \leq P'$ processors and n initial data in time $T^p(P, N) \leq \frac{c'_p \pi n}{P' - kc'_p \pi n}$, with $\pi = \frac{P'+P}{P}$.*

Note that P, P' are in general functions of the problem size. Hence, when A' is rescheduled on n'' data, we have to enter $P' = P'(n'')$ in the bound Proposition 2. Since relation (6) can be rewritten as $t'_p = \frac{c'_p n}{P' - kc'_p n}$, the upper bound of the proposition can be interpreted as the time needed by the original algorithm A' working on πn initial data. This time can be made arbitrarily large for proper values of n, while t'_p stays limited, thereby making relation (2) non significant. Proposition 2 can be extended to different complexity functions with foreseeable results. An example is shown in the next section.

Let us now consider a *census* problem with time-varying data, solved under Paradigm 1. The computation is completed when all data currently arrived have been processed, independently of other data that may arrive at later times.

Problem 1. (1) Given a set I of integers, of size $N = n + knt$, compute the sum S of all the elements in I.

Problem 1 occurs in an idealized banking operation, where the balance S of all movements in an account must be computed before a withdraw is honored. Note that the number of movements is given by an initial value n, plus an increment proportional to time and n. When S has been found, the algorithm terminates, and a different operation (e.g, a paying procedure) is initiated. For $n < \frac{1}{kc}$ this problem can be solved with a sequential linear time d-algorithm consisting of a scan, according to relation (4) with $\alpha = 1$. As well known, it can be solved in parallel on the initial n data with $P = \frac{n}{\log n}$ processors in time $O(\log n)$. For time-varying data we have:

Proposition 3. *Problem 1 admits a pd-algorithm with* $P = \frac{n}{\log n}$ *and* $t_p \leq c_p \log n$.

The algorithm is as follows:

1) subdivide the initial set of n data in P sections $I_1, \ldots, I_P$, of $\log n$ elements each; each processor computes the sum in I_i sequentially in time $c \log n$;
2) during step 1, $ckn \log n$ new data arrive. Assuming that $n \leq \frac{1}{ck}$, we have $ckn \log n \leq \log n$, then, we still have to compute the sum of $\leq \frac{n}{\log n} + \log n$ elements. This sum is computed in a binary tree fashion in time $c \log n$;
3) during step 2, $ckn \log n \leq \log n$ new data arrive. We can compute the sum of these data, plus the result of step 2, in time $c \log \log n$. The computation is iterated on the $\log \log n$, $\log \log \log n, \ldots, 1$ data arrived in the previous steps.

We have $t_p = c(\log n + \log n + \log \log n + \ldots) \leq c_p \log n$.

In the above algorithm, we have assumed an upper bound $\frac{1}{ck}$ on n. Since this value coincides with the asymptote of the sequential case, we can make a fair estimate of the parallel speed-up. Proposition 1 can be reformulated for $N = n + knt$ and $P = \frac{n}{\log n}$, to state that the ratio $\frac{t}{Pt_p}$ tends to ∞ for n tending to $\frac{1}{ck}$, hence, the above is a pd-algorithm. In fact, we have $t = \frac{cn}{1-ckn}$ and $t_p \leq c_p \log n$, hence $\frac{t}{Pt_p} \geq \frac{c}{c_p(1-ckn)}$. Proposition 2 can also be adapted to Problem 1, showing that Brent's principle can be violated in absolute and asymptotic terms.

3 Computing with faulty processors (Paradigm 2)

The new paradigm of parallel computation considered here is based on the assumption that the PRAM processors may fail. Upon occurrence of a failure, a processor stops permanently. Therefore, the algorithm must be organized in such a way that, for any processor failure, all the pending operations can still be executed.

To make a PRAM program robust, a common target is *graceful degradation*, that is, the computation is correctly carried to an end as long as at least one processor survives [4, 6, 8]. For this purpose, the operations must be dynamically assigned to the processors still alive, with some unavoidable replication. The goal is then the minimization of the total number of operations performed, while no bound can be posed on the total running time.

Our paradigm is completely different. Assuming a known probability distribution of the processor failures, we are able to set an upper bound on the running time, with a given probability of successful termination. Indeed, bounding the running time is a common requirement in algorithm design, and is crucial in real-time systems (see section 2) and in other practical contexts. Our result is obtained at the expense of an increase of the total number of operations, due to the need of their replication over the original *fault intolerant* algorithm (i.e., a

standard algorithm designed for non failing processors). As anticipated in section 1, the bounds on parallel speed-up must be revised. In particular, the scaling bound of relation (2) is modified in an intrigued and nontrivial way.

Our paradigm is as follows. Starting from any fault intolerant PRAM algorithm, we increase the total number of processors, and allocate a subset S of processors to each operation O with a schedule decided off-line. The operation O is executed concurrently by all the processors in S which are still alive (this requires a COMMON CRCW-PRAM variant). The cardinality of S is chosen to guarantee that the operation is completed with a given probability.

Let p be the probability that an arbitrary processor E_j completes the t-th step of its program, given that it has completed all the previous steps (shortly, E_j is *alive* when entering step t). We assume that p is a constant with respect to t and is the same for all processors. Formally, for any processor index j and time step i, define the event

$$C_i^j = \text{“Processor } E_j \text{ completes the } i\text{-th step of its program”},$$

and, for any $t > 0$ and any given j, let

$$\Pr\left(C_t^j \mid \bigcap_{i=1}^{t-1} C_i^j\right) = p. \tag{9}$$

As an additional condition, we assume that distinct processors behave independently, that is, for any choice of indices (i, j) and (h, k) with $j \neq k$, the events C_i^j and C_h^k are independent. Note that relation (9) captures the common assumption on the exponential distribution of hardware failures in a model with discrete time [10].

Consider any fault intolerant PRAM algorithm A running in time T and performing a total number X of operations. At each step t, the algorithm performs the operations O_t^k, $1 \leq k \leq X_t$, $\sum_{t=1}^{T} X_t = X$. We will allocate a *cluster* S_t^k of processors to each operation O_t^k, so that the operation is executed with at least a fixed constant probability $\bar{p}$. For $\bar{p}$ to be constant, the size of S_t^k must increase with t, because the probability that a processor in S_t^k is still alive decreases with t.

The value of $\bar{p}$ clearly influences the overall probability that the new fault tolerant algorithm $\bar{A}$ is completed successfully. Let $C_{\bar{A}}$ be the event “algorithm $\bar{A}$ is completed successfully”. For $1 \leq t \leq T, 1 \leq k \leq X_t$, define the event:

$$I_t^k = \text{“No processor in } S_t^k \text{ completes } O_t^k\text{”}.$$

We have $\Pr(I_t^k) \leq 1 - \bar{p}$, therefore

$$\Pr(C_{\bar{A}}) = 1 - \Pr\left(\bigcup_{t=1}^{T} \bigcup_{k=1}^{X_t} I_t^k\right) \geq 1 - X(1 - \bar{p}) \tag{10}$$

We require that $\Pr(C_{\bar{A}}) \geq r$, for fixed r. Then it suffices that $\bar{p}$ satisfies the relation:

$$1 - X(1 - \bar{p}) \geq r. \tag{11}$$

We are now ready to determine the size s_t^k of the clusters S_t^k. For this purpose, define the event:

Y_t^k = "at least one of the processors in S_t^k is alive after the first t steps".

By unfolding relation (9), we have that the probability that an arbitrary processor completes the first t steps of its program is p^t, therefore:

$$\Pr(Y_t^k) = 1 - \Pr\left(\bigcap_{h=1}^{s_t^k} \overline{\bigcap_{j=1}^{t} C_t^h}\right) = 1 - \left(1 - p^t\right)^{s_t^k}.$$

We require that, for any t and k, $\Pr(Y_t^k) \geq \bar{p}$. Then it suffices that s_t^k be such that $1 - (1 - p^t)^{s_t^k} \geq \bar{p}$, whence

$$s_t^k \geq \frac{\log_e(1 - \bar{p})}{\log_e(1 - p^t)}. \tag{12}$$

Note that the right hand side of relation (12) is independent of k. Hence all the s_t^k, $1 \leq k \leq X_i$, can be set to the same value s_t. It follows that the number of processors s required to complete algorithm $\bar{A}$ with probability at least r is

$$s = \max\{X_t s_t, 1 \leq t \leq T\}. \tag{13}$$

The above results can be applied to the solution of any problem in Paradigm 2. As an example, let us apply the paradigm to solve Problem 1 of section 2 with constant data size N, as a framework for any census problem. The problem can be solved on a P-processor PRAM, $1 \leq P \leq N$, by a uniform family of fault intolerant parallel algorithms $\{A_P\}$, whose computation is organized in two phases. In the first phase, the initial set of data is subdivided into P disjoint sections, one for each processor, and the sum in each section is computed sequentially in time $\lceil \frac{N}{P} \rceil - 1 \leq \frac{N}{P}$. In the second phase, the processors perform a tree-like computation on the P partial results in time $\lceil \log P \rceil - 1 \leq \log P$. The overall time requirement of A_P is $\frac{N}{P} + \log P$, while the total number of operations, including the ones of idling processors, is $X = N + P$.

If the processors may fail, each algorithm A_P must be transformed into a corresponding fault tolerant algorithm $\bar{A}_P$. Recall that A_P and $\bar{A}_P$ require the same time, since the operations of $\bar{A}_P$ are the same as in A_P. However, each operation O_t^k is executed in $\bar{A}_P$ in parallel by all the processors still alive in cluster S_t^k. In the framework of Paradigm 2, $\bar{A}_P$ requires a number $s(P)$ of processors given in relation (13). Through relations (11) and (12), we have that $s(P)$ depends on the required probability r of successful completion. As customary in randomized algorithms, we set $r \geq 1 - \frac{1}{N^c}$, for a given constant c, independently of the value of P.

From relation (11) we have $1 - X(1 - \bar{p}) = 1 - (N + P)(1 - \bar{p}) \geq 1 - \frac{1}{N^c}$, whence

$$\bar{p} \geq 1 - \frac{1}{N^c(N + P)}. \tag{14}$$

From relation (12) we then derive the number of processors $s_t^k = s_t$ for operation O_t^k, namely:

$$s_t = \frac{\log_e(N^c(N+P))}{\log_e \frac{1}{1-p^t}}. \tag{15}$$

Given that $\log_e \frac{1}{1-p^t} \geq p^t$ for any $t > 0$, and recalling that $P \leq N$, we have $s_t \leq (c+2)\log_e N \left(\frac{1}{p}\right)^t$. To apply relation (13) we still have to determine the values of X_t for the two phases of the algorithm. We have $X_t = P$, $1 \leq t \leq \frac{N}{P}$ (first phase), and $X_{N/P+i} = \frac{P}{2^i}$, $1 \leq i \leq \log P$. Therefore:

$$s(P) \leq \max \left\{ \begin{array}{ll} (c+2)P \left(\frac{1}{p}\right)^t \log_e N, & 1 \leq t \leq \frac{N}{P} \\ (c+2)P \left(\frac{1}{p}\right)^{\frac{N}{P}} \left(\frac{1}{2p}\right)^i \log_e N, & 1 \leq i \leq \log P \end{array} \right\} \tag{16}$$

From relation (16), we derive with easy calculations:

Proposition 4. *For any $p > \frac{1}{2}, c > 0$ and $P \leq N$, Problem 1 with constant data size N admits a family of fault tolerant algorithms with success probability $\geq 1 - \frac{1}{N^c}$, requiring time $\frac{N}{P} + \log P$ and a number of processors $s(P) \leq (c+2)P \left(\frac{1}{p}\right)^{N/P} \log_e N$.*

Note that the assumption $p > \frac{1}{2}$ is plausible, since p is the probability that a processor completes a single step. For example, for $P = N$, algorithm $\bar{A}_P$ solves the problem in time $\log N$ with probability $\geq 1 - \frac{1}{N^c}$, with $O(cN \log N)$ processors.

Proposition 4 puts into evidence an unexpected property of our paradigm. As the number of processors P of the fault intolerant algorithm A_P increases, hence the running time T of A_P and $\bar{A}_P$ decreases, the number of processors $s(P)$ of $\bar{A}_P$ may increase or decrease for proper values of p and N. Hence, the total work $s(P)T$ may decrease for increasing P, against all the bounds on parallel speed-up. For example note that, for fixed p, we have $s(\frac{N}{\log_{1/p} N}) = \Omega(cN^2)$, from (12) and (14), and $s(N) = O(cN \log N)$ from Proposition 4, that is $s(N) = o(s(\frac{N}{\log N}))$.

In fact, formula (1) should now express a $\leq$ relation between $s(P)T(P,N)$ and $s(1)T(1,N)$, which can hold in any direction for proper values of p and N (note that the fault tolerant version $\bar{A}_1$ of a sequential algorithm A_1, requires $s(1) > 1$ processors). Against a fair expectation from relation (1), we have that $s(P)T(P,N) \leq s(1)T(1,N)$ for $P \to N \to \infty$ and fixed p, thereby proving the inherent power of the parallel solution.

4 Computing in a secure environment (Paradigm 3)

The theory of computing relies on the assumption that computable problems can be solved sequentially. There are situations, however, where the intrinsic power

of parallel computation may be necessary. A classical request is that an action be performed by several distinct agents to certify the result reliably. A similar request may be raised for security reasons. Different schemes of computation thus arising are grouped in Paradigm 3.

For a given computation, consider a *Data Dependency DAG* (DDD) of V vertices, where vertices correspond to a single operations and edges specify data dependencies [1]. The set of vertices is partitioned into *strata* $S_0, \ldots, S_{h-1}$, such that u is in S_i if i is the length of the longest path from a source to u (source vertices are in S_0). The resulting structure is called SDDD. Let $P \geq 1$ processors be available, and assume that each operation can be performed by any processor in unit time. A *scheduling* of SDDD is an assignment of a processor $q(u)$ and a time $t(u)$ to each vertex u such that:

1. $q(u) \neq q(v)$ for $t(u) = t(v)$;
2. $t(u) < t(v)$ for $u \in S_i$, $v \in S_j$, $i < j$.

If $P \geq \max\{|S_i|, 0 \leq i \leq h-1\}$, there is a straightforward scheduling with $t(u) = i$ for each $u \in S_i$, $0 \leq i \leq h-1$. The *duration* of the scheduling (i.e., the time required by the computation) is h and is obviously optimal. For $P < \max\{|S_i|, 0 \leq i \leq h-1\}$, each stratum S_i can be divided into $\left\lceil \frac{|S_i|}{P} \right\rceil$ new strata, and the above scheduling applied to the resulting SDDD. The duration is then $\leq \frac{V}{P} + h$, distant from optimal within a factor of 2.

We now introduce our new paradigm. Let $Q = \{q_1, \ldots, q_P\}$ be the set of processors, and $\{v_1, \ldots, v_s\}$ be an arbitrary stratum of SDDD. Any single operation v_i must be executed by a subset of processors, according to the following design rules:

A1: an integer n_i, $0 < n_i \leq P$, is assigned to v_i, with the intention that n_i arbitrary processors must be employed;
A2: a fixed subset $Q_i \subseteq Q$ is specified for v_i;
B1: the processors assigned to v_i may operate at different times;
B2: all the processors assigned to v_i must operate at the same time.

We construct a scheduling of SDDD by scheduling the operations of each stratum $S = \{v_1, \ldots, v_s\}$ independently of the other strata, under the following combinations of the above rules. Let d be the number of parallel steps required for S:

A1,B1 *Easy.* An optimal scheduling is trivially attained by a linear time assignment procedure. We have $d = \left\lceil \sum_{i=1,s} n_i \, / P \right\rceil$.

A1,B2 *Difficult.* The scheduling is equivalent to bin-packing, where the n_i's are to be packed into bins of size P. Applying known bin-packing heuristics we attain $d < \left\lceil 2 \sum_{i=1,s} n_i \, / P \right\rceil$, and d is within a constant factor from optimal.

A2,B1 *Easy.* The scheduling can be reformulated as a sequence of max-flow problems on a bipartite graph built on the sets S and Q, with edges $[v_i, q_j]$ for $q_j \in Q_i$, augmented with a source and a sink vertex. However, this max-flow problems are particularly easy, and can be collectively solved in time

$\sum_{i=1,s} |Q_i|$, by a linear scan of the Q_i's. We have $d = \max\{k_i, 1 \leq i \leq P\}$, where k_i is the number of occurrences of q_i in $Q_1, \ldots, Q_s$. Note that d is clearly optimum.

A2,B2 *Open.* The scheduling of S reduces to an Open Shop Scheduling with tasks of length 0 or 1 [3]. A (non evaluated) heuristic can be constructed as an extension of the First Fit heuristic for bin-packing, with proper additional constraints on the selection of the subsets of processors to be allocated in each bin.

5 Conclusion and extensions

The main goal of this paper is to stimulate a discussion on parallel algorithm design in non conventional situations, where the increase in the number of processors may result in a drastic improvement of performance. We have restricted our analysis to three computational paradigms. Variations and alternatives to our schemes are possible and desirable. Referring to Paradigm 1, we may assume that a problem has fixed data size, but variations of the data values occur with time. An algorithm must now be capable of changing the effect of its previous operations which involve a datum D, if D is later modified. A theory similar to the one for d-algorithms can be developed.

Paradigm 1 is unsatisfactory if the data arrived after completion of the algorithm cannot be ignored. Rather, the stream of data must be considered without an end, as required for example in operating systems and data base maintenance [7]. The time can be divided in slots of length T. Each slot consists of an updating phase, during which the data accumulated in the previous slot are treated, followed by a free phase where other routines may be run. If t and t_p are the sequential and parallel times to solve a problem, define the free time gain $G = \frac{T-t_p}{T-t}$. Unexpectedly G may assume any value between 1 and ∞ for certain families of problems and for proper values of the parameters.

Several variations of Paradigm 2 are significant. In particular, we may consider an on-line allocation strategy such that, at each time step, the subset of processors assigned to each operation is chosen at random, or is taken deterministically from the set of processors that are still alive [8]. The analysis of these cases, and the implications on the bounds of parallel speed-up, are challenging open problems.

Finally, Paradigm 3 is centered on scheduling problems, which have been approached independently on single SDDD strata. A global optimization strategy on SDDD as a whole is likeley to yield better results.

References

[1] A. Aggarwal and A.K. Chandra. Communication Complexity of PRAMs. *Proc. 15th Int. Colloquium on Automata, Languages and Programming* (1988) 1-18.

[2] S.G. Akl, M. Cosnard and A.G. Ferreira. Data-movement-intensive problems: two folk theorems in parallel computation revisited. *Theoretical Computer Science* **95** (1992) 323-337.

[3] M.R. Garey and D.S. Johnson. *Computers and Intractability.* Freeman, San Francisco, 1978.

[4] P.C. Kanellakis and A.A. Shvartsman. Efficient parallel algorithms can be made robust. In *Proc. 8th Annual ACM Symp. on Principles of Distributed Computing* (1989) 211-222.

[5] R.M. Karp and V. Ramachandran A survey of parallel algorithms for shared memory machines. In *Handbook of Theoretical Computer Science* North Holland, New York NY (1990) 869-941.

[6] Z.M. Kedem, K.V. Palem and P.G. Spirakis. Efficient robust parallel computations. In *Proc. 22nd Annual ACM Symp. on Theory of Computing* (1990) 590-599.

[7] F. Luccio and L. Pagli. *The p-shovelers problem. (Computing with time-varying data). SIGACT News* **23**, 2 (1992) 72-75

[8] C. Martel, R. Subramonian, A. Park. Asynchronous PRAMs are (almost) as good as synchronous prams. In *Proc. 31st Symp. on Foundations of Computer Science* (1990) 590-599.

[9] W. Paul. On line simulation of $k+1$ tapes by k tapes requires nonlinear time. *Information and Control* **53** (1982) 1-8.

[10] K.S. Trivedi. *Probability and statistics with reliability, queueing, and computer science applications.* Prentice-Hall, Englewood Cliffs NJ (1982).

[11] U. Vishkin. Can parallel algorithms enhance serial implementation?. *SIGACT News* **22**, 4 (1991) 63.