Modeling the Benefits of Mixed Data and Task Parallelism

Soumen Chakrabarti* James Demmel† Katherine Yelick*

Abstract

Mixed task and data parallelism exists naturally in many applications, but utilizing it may require sophisticated scheduling algorithms and software support. Recently, significant research effort has been applied to exploiting mixed parallelism in both theory and systems communities. In this paper, we ask how much mixed parallelism will improve performance *in practice*, and how architectural evolution impacts these estimates. First, we build and validate a performance model for a class of mixed task and data parallel problems based on machine and problem parameters. Second, we use this model to estimate the gains from mixed parallelism for some scientific applications on current machines. This quantifies our intuition that mixed parallelism is best when either communication is slow or the number of processors is large. Third, we show that, for balanced divide and conquer trees, a simple one-time switch between data and task parallelism gets most of the benefit of general mixed parallelism. Fourth, we establish upper bounds to the benefits of mixed parallelism for irregular task graphs. Apart from these detailed analyses, we provide a framework in which other applications and machines can be evaluated.

1 Introduction

Mixed parallelism exists naturally in many applications. In adaptive mesh refinement (AMR) algorithms, there is task parallelism between meshes and data-parallelism within a mesh [2]. In computing eigenvalues of nonsymmetric matrices, the sign function algorithm does divide and conquer with matrix factorizations at each division [3]. In timing-level circuit simulation there is parallelism between separate subcircuits and parallelism within the model evaluation of each subcircuit [26]. In sparse matrix factorization, multi-frontal algorithms expose task parallelism between separate dense sub-matrices and data parallelism within those dense matrices [16]. In global climate modeling [19], there are large data parallel computations performed on grids representing the earth's atmosphere and oceans, and task parallelism from the different physical processes being modeled.

Several researchers have proposed support to take advantage of this mixed parallelism. In the theory area, the best known on-line scheduling algorithm for mixed parallelism is 2.62-optimal [4, 11], and the best off-line algorithm is 2-optimal [25, 17]. In the systems area, the Paradigm compiler [20], iWarp compiler [24], and NESL compiler [6] all support limited forms of mixed task and data parallelism, and there are plans to merge data Fortran D with Fortran M [12] and pC++ with CC++ [15] to support mixed parallelism.

In this paper, we step back from these algorithmic and systems issues and address the question of how much benefit should be expected, and what impact architectural evolution has on these estimates. Specifically, we consider the relative efficiency of executing a task graph with parallelizable tasks using mixed parallelism vs. pure data parallelism. In a purely data parallel execution, the tasks in the task graph are executed one at time using all the processors for each. In mixed parallelism, each task is spread over a subset of processors. We are generous in our treatment of mixed parallelism in that tasks are modeled as having no setup or switching cost, and optimal scheduling is assumed. By using a performance model derived from a variety of numerical applications, we conclude that the speedup of mixed over data parallelism is often modest, depending on the application and machine. Furthermore, much of the efficiency of mixed parallelism can be achieved with a much simpler scheduling strategy which we call *switched parallelism*, in which either data or task parallelism is used at a given time. Switched parallelism has been found empirically useful in FFT, sorting and certain eigenvalue solvers. Here we analyze how close switched parallelism can be to optimal efficiency.

Our modeling approach has two parts. First, we model the efficiency profile of a single parallelizable task. Second, we introduce simple forms of task parallelism and see how well this supplements the data parallelism within the tasks. For both cases we provide analytical and experimental estimates for the maximum possible performance gains obtained using mixed parallelism. In doing this, we make various reasonable assumptions about the tasks and the task graph. Our modeling approach may be of independent interest, because the basic recipe can easily accommodate different model components.

The paper is organized as follows. Section 2 presents and justifies our model in the context of our problem domain. In Section 3, we estimate the performance benefits of mixed parallelism for a batch of independent tasks. We show that over all possible batches with various task sizes, a balanced batch with identical tasks shows off mixed parallelism to the greatest potential benefit, so it suffices to study this case in detail. Next we consider task graphs with dependencies in Section 4. For balanced divide and conquer trees, we present simulation estimates of data parallel, switched parallel, and

*Computer Science Division, U. C. Berkeley, CA 94720. Supported in part by ARPA/DOD (DABT63-92-C-0026), DOE (DE-FG03-94ER25206), and NSF (CCR-9210260, CDA-8722788 and CDA-9401156). The information presented here does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred. Email: {soumen,yelick}@cs.berkeley.edu

†Computer Science Division and Mathematics Department, U. C. Berkeley, CA 94720. Supported in part by NSF (ASC-9005933, CDA-9401156), ARPA contract DAAL03-91-C-0047 via a subcontract from the University of Tennessee, ARPA grant DM28E04120 via a subcontract from Argonne National Laboratory, and DOE grant DE-FG03-94ER25206. Email: demmel@cs.berkeley.edu

mixed parallel execution efficiencies. For irregular graphs, we get some general but relatively weaker upper bounds on the maximum marginal improvement by mixed parallelism over data parallelism. In Section 5 we present experience with a recent mixed parallel application. Section 6 suggests extensions and Section 7 draws conclusions.

2 The model

The performance gain from using mixed parallelism instead of pure data or task parallelism is a function of the machine parameters and the workload, which we visualize as a task graph with vertices representing parallelizable tasks. The parallel subroutines represented by these vertices, together with machine parameters like network latency and bandwidth, define how scalable these parallelizable tasks are. We thus reduce the factors affecting performance to the following.

1. The scalability (equivalently, efficiency or speedup) profile of the parallelizable tasks. Throughout this paper, all vertices are assumed to run the same parallel subroutine (or have the same efficiency profile) but on different problem sizes. This is a reasonable model for a variety of divide and conquer problems.
2. The structure of the task graph, which gives an idea of the degree of task parallelism available to supplement data parallelism. By "structure" we mean the task vertices and directed precedence edges of the graph and the problem sizes at the vertices.

Accordingly, our model has two components. In Section 2.1 we model a single task profile, and in Section 2.2 we model the task graph.

2.1 The efficiency of data parallelism

We let $e(N,P)$ be the parallel efficiency of solving a problem of size N on P processors. If the serial running time is $f(N)$, the parallel running time $r(N,P)$ on P processors is $r(N,P) = f(N)/(P \cdot e(N,P))$. $e(N,P)$ depends on the algorithm, and relative speeds of computation and communication. Despite e's possibly complex dependence on all these parameters, we will show that for a number of algorithms of interest, $e(N,P)$ is accurately modeled by a simple two-parameter function of the problem size per processor, N/P.

By Amdahl's law, we expect e to be a decreasing function of P, with $e(\cdot,1) = 1$. So our intuition is that $e(N,P)$ should be an increasing function of N/P. We will let $e_\infty \leq 1$ be its asymptotic value for large N/P. The next question is how $e(N,P)$ approaches e_∞. There are, of course, an infinite number of functions to model this, but we shall propose a simple model that we will empirically validate. Roughly speaking, the model captures programs having an area-to-volume relationship between communication and computation, which abounds in parallel scientific applications.

2.1.1 The asymptotic model

The efficiency of a data parallel task of size N on P processors is modeled as

$$e(N,P) = \begin{cases} 1 & \text{if } P = 1 \\ \dfrac{e_\infty}{1 + \sigma P/N} & \text{if } P > 1. \end{cases} \quad (1)$$

The parameter σ measures how fast the efficiency approaches its asymptotic value e_∞. As shown in figure 1 the efficiency reaches half its asymptotic value when $N/P = \sigma$. Thus, the smaller the value of σ, the more efficient the implementation is for a fixed problem size. The parallel running time $r(N,P)$ is

$$r(N,P) = \frac{f(N)}{e_\infty}\left(\frac{1}{P} + \frac{\sigma}{N}\right). \quad (2)$$

Equation (2) says that adding processors has diminishing returns, much like Amdahl's law. However, since no sequential and perfectly parallel components can be identified, the asymptotic model is not identical to Amdahl's law.

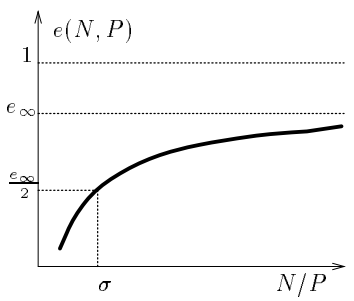

Figure 1: The proposed efficiency model for data parallelism within a single task.

2.1.2 Validation

We validated our model using experimental data. In figure 2, we consider three ScaLAPACK programs: LU, QR and Cholesky factorizations, and three machines: the Delta, Paragon and iPSC/860 [8]. Each graph plots performance in GFLOPS per processor versus N/P, including experimental data (the circles), as well as the prediction of the asymptotic model. The iPSC/860 experiments were run with 128 processors, and the Paragon and Delta experiments were run with both 128 and 512 processors. Each graph includes an estimate s_inf of the per-processor GFLOPS as $N/P \to \infty$ and an estimate of σ (sigma). The asymptotic model is a good fit for the actual efficiency profiles: the mean relative error is 6–11%.

Estimates of σ are important for performance analysis as well as runtime scheduling decisions, as we shall see later. To this end, we collect values of σ for some parallel scientific libraries [8], using existing analytical performance models [9, 10]. For each of these routines, we have available the communication and computation time as functions of problem size, number of processors, network latency, and network bandwidth. Using these given functions, we first estimate the parallel running time $r(N,P)$ for a given machine and problem, then fit Equation (1) to it. The results are presented in Table 1.

2.2 Task graph model

The second part of our model has to address the task graph structure. In the theory literature, irregular and even online task graphs are handled, but the algorithms are optimal in the asymptotic sense with constants in the range 2–2.6, in the worst case. Unfortunately, a constant factor

Machine	α	β	M/P	σ_{MM}	σ_{LU}	σ_{BS}	σ_{SF}
Alpha+ATM1	3.8×10^5	62	64	1.3×10^4	5.7×10^6	3.4×10^6	2.7×10^6
Alpha+ATM2	3.8×10^5	15	64	6500	5.6×10^6	3.4×10^6	2.7×10^6
Alpha+Ether	3.8×10^5	960	64	2.5×10^5	6.9×10^6	4.2×10^6	3.4×10^6
Alpha+FDDI	3.8×10^5	213	64	4.1×10^4	5.9×10^6	3.6×10^6	2.9×10^6
CM5	450	4	32	53	490	2234	3826
CM5+VU	1.4×10^4	103	32	9100	3.1×10^5	1.9×10^5	1.53×10^5
Delta	4650	87	16	7400	1.5×10^5	9.3×10^4	7.2×10^4
HPAM (FDDI)	300	13	64	154	9300	5400	4250
iPSC/860	5486	74	16	5490	1.5×10^5	9.2×10^4	7.3×10^4
Paragon	7800	9	16	633	1.25×10^5	7.7×10^4	6×10^4
SP1	2.8×10^4	50	64	4250	4.8×10^5	2.9×10^5	2.4×10^5
T3D	2.7×10^4	9	64	1544	4.2×10^5	2.5×10^5	2×10^5

Table 1: Estimates of σ for different machines and problems in the asymptotic model. The Alphas use PVM as messaging software. ATM1 = current generation; ATM2 = projected next generation. HPAM = a cluster of HP workstations connected by FDDI with a prototype active message implementation. The programs are matrix multiplication (MM), LU factorization (LU), backsolve (BS), and sign function (SF, discussed later). $e_\infty = 1$ for all problems in this table. Parameters α (latency) and β (inverse bandwidth) are normalized to a BLAS-3 FLOP, and the model is fit to data generated from analytical models [9, 10, 23]. The curves were fit for $2 \leq P \leq 500$ and $100 \leq n = N^{1/2} \leq 10000$. An estimate of memory per processor in megabytes is given in the column marked M/P. Estimates for α and β are in part from [23, 27, 18, 1].

of this magnitude (which we will call "packing loss") may substantially mask the benefits which would otherwise be obtained from mixed parallelism. Furthermore, we know of no tighter analysis of this constant for a given graph. We are thus faced with the following problem. Data parallelism is easy to load balance and schedule, but has scalability limits (expressed by our $e(N,P)$ model). Task parallelism has ideal efficiency but shows load imbalance.

We circumvent this problem by considering how mixed parallelism will perform in very favorable circumstances, namely in regular divide and conquer trees. Our assumptions are listed below.

- The task graph is a complete tree with branching factor $d \geq 2$.
- The d child tasks of a task of size N are all of size N/c, where $c > 1$.
- The work required to do a task of size N is $f(N) = N^a$, where $a \geq 1$.

We call such regular trees that have a root size of N as (N, a, c, d) trees. This is a very simple model, and so we need to understand the limits of its applicability. First, task communication is not accounted for. But for $a > 1$, task communication cost is generally of lower order than the node cost $f(N)$ (e.g., $O(N)$ vs. $O(N^{3/2})$ in the case of many dense matrix operations). Thus, we expect our model to overestimate the benefits of mixed parallelism, provided problems are large enough. Therefore, a prediction of little benefit from mixed parallelism for a particular problem is likely to be trustworthy, while a prediction of great benefit from mixed parallelism must be further analyzed. Second, for evaluating efficiency gains with high accuracy, only regular trees could be considered.

In Section 4.2 we develop a comparatively weak bound to the benefits of mixed parallelism for irregular graphs. In spite of the above restriction, we demonstrate interesting effects of the tree shape and size on the optimal scheduling strategy.

3 Batch problems

We will use the efficiency models of the last section to determine the best way to allocate processors to a single task, and then to a batch of L independent, identical tasks. We argue that the benefit of mixed over data parallelism is largest when the independent tasks are identical, rather than being of different sizes. Finally, we give a simple near-optimal heuristic for switched execution of a batch of tasks of various sizes.

3.1 Balanced batch problems

For a single task with sequential running time $f(N)$, the choice is only between 1 and P processors, and the running time is

$$f(N) \times \min\left\{\frac{1}{e_\infty}\left(\frac{1}{P} + \frac{\sigma}{N}\right),\ 1\right\}$$

For a batch of L independent tasks, each of size N, the sequential running time t_1 of all L tasks is $Lf(N)$. The data parallel running time t_D, where we run each task in data parallel fashion one after the other, is just L times the above expression. We let t_T denote the task parallel running time, where we assign one processor per task. Finally, we let t_M denote the mixed parallel running time, the optimal running time over all possible assignments of processors to tasks. Let e_D, e_S, and e_M be the corresponding overall efficiencies.

By allocating only one processor per task, we can get parallel execution time $\lceil L/P \rceil f(N)$. Since the work lower bound is $Lf(N)/P$, pure task parallelism is optimal when $L \geq P$ (modulo rounding effects, which we ignore here and elsewhere). Thus, we need only consider the case $L < P$. Also assume L divides P. The following is easily seen.

Lemma 3.1 *When L divides P, the running time for L independent tasks of size N in the asymptotic model using optimal mixed parallelism is $t_M = f(N) \times \min\left\{1, \frac{1}{e_\infty}\left(\frac{L}{P} + \frac{\sigma}{N}\right)\right\}$, and the running time using data parallelism is $t_D = Lf(N) \times \min\left\{1, \frac{1}{e_\infty}\left(\frac{1}{P} + \frac{\sigma}{N}\right)\right\}$.*

The next corollary says how much faster mixed parallelism can be than data parallelism.

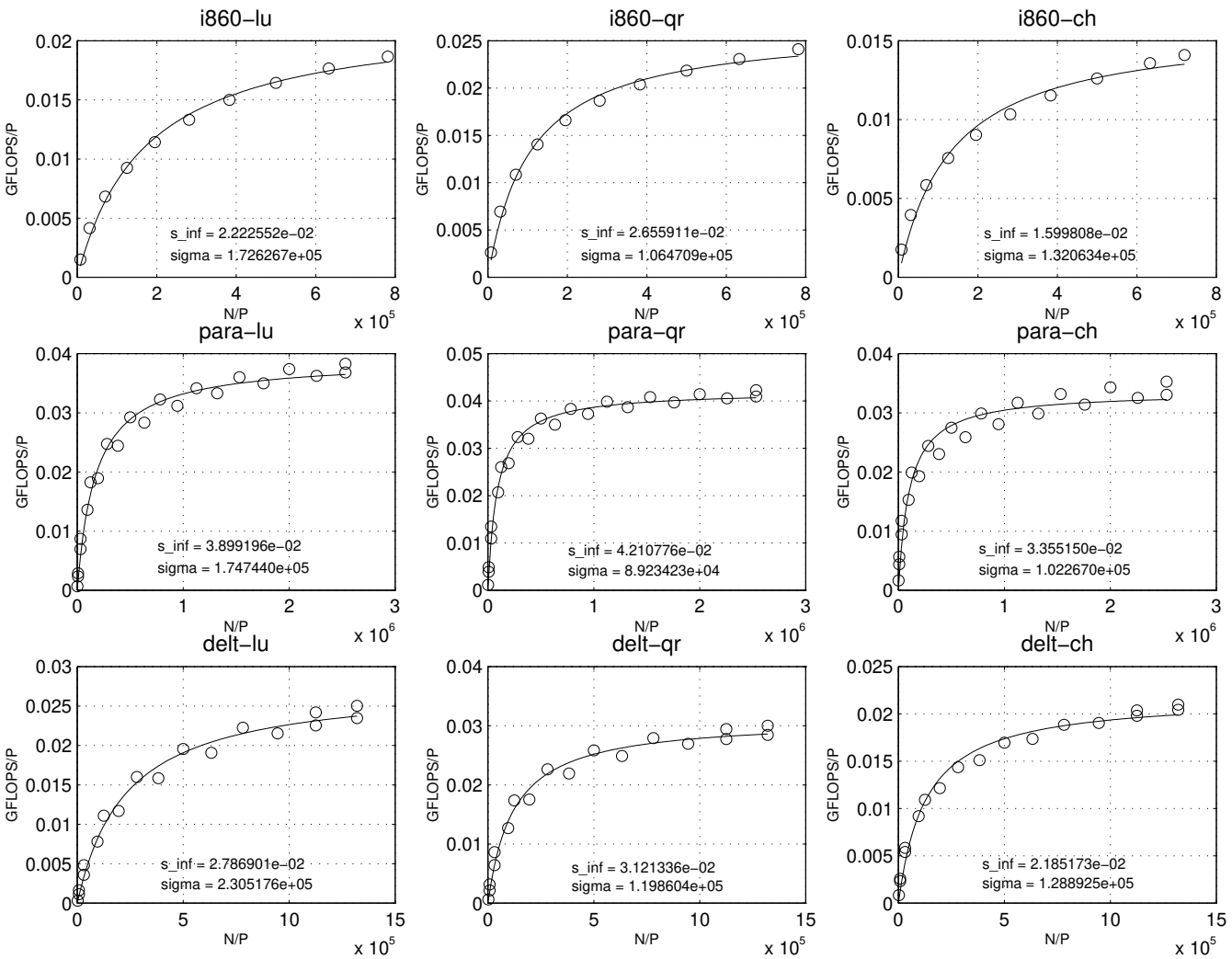

Figure 2: Validation of the asymptotic model using data from the ScaLAPACK implementations of LU, QR, and Cholesky (CH) factorization programs. The machines are iPSC/860 (i860), Paragon (para) and Delta (delt). In each graph, s_inf is the per-processor GFLOPs as $N/P \to \infty$, and sigma is the σ in the asymptotic model.

Corollary 3.2 *When task parallelism is not optimal, the relative improvement of using mixed parallelism over pure data parallelism for the batch problem is*

$$\frac{e_M}{e_D} = \frac{\frac{N}{\sigma P} + 1}{\frac{N}{\sigma P} + \frac{1}{L}} \tag{3}$$

Example 3.3 We apply this analysis to complex matrix multiplication, which was reported as a benchmark for the Illinois Paradigm compiler [20]. The task is MM, the machine is the CM5 without vector units, and $L = 4$. From figure 1 we obtain $\sigma = 53$ and $e_\infty = 1$ for this problem. To attain a relative improvement ϵ of mixed over data parallelism, i.e. $e_D/e_M < 1 - \epsilon$, we need the problem size to be small. Specifically, if each MM involves $n \times n$ matrices, we can substitute the numbers into (3) and see that n needs to be less than roughly $\sqrt{53P(3-4\epsilon)/4\epsilon}$ for this improvement. E.g., if $P = 64$ and $\epsilon = 0.5$, then $n < 42$, a tiny problem indeed. It is interesting that the experiments reported in [20] for the CM5 use $P \in \{64, 128\}$ processors and $n = 64$. On the Paragon with $P = 512$ and $\sigma = 633$, to ensure $e_D/e_M < 0.5$ as above, we will need roughly that $n < 569$, which is still not large by the standard of many scientific applications: a matrix of this size fills only 0.03% of the Paragon's total memory. □

The conclusion is that MM data parallelizes too well to benefit much from mixed parallelism on a machine as balanced (i.e. with as low a β) as the CM5 without vectors units. Better hunting grounds for cases where mixed parallelism helps significantly are more unbalanced machines (high β) and problems with less scalable data parallel components.

Example 3.4 Changing the problem to BS in the last example and changing the machine to a 16 processor SP1, we see that $n < 1386$ must hold for $e_D/e_M \leq 0.5$. This is a relatively realistic size. Mixed parallelism improves the efficiency from roughly 33% to 66%. □

3.2 Unbalanced batch problems

So far we have considered batches of tasks of identical size N. Here we argue that permitting tasks of different sizes makes data parallelism only closer in performance to optimal mixed parallelism, because, if there are a few large tasks that dominate the work content of the batch, the margin for improvement over pure data parallelism will be small.

To prove this, suppose we have a batch of $L > 1$ tasks, the i-th task of size N_i. Suppose all tasks have the same efficiency profile $e(N_i, P)$, and that the sequential processing time for task i is $f(N_i)$, where $f(x) = x^a$ as before. We will need the following theorem for our proof.

Theorem 3.5 (Hölder's Inequality) *Let $x_k, y_k > 0$ for $1 \le k \le L$, $p > 1$, and q be such that $\frac{1}{p} + \frac{1}{q} = 1$. Then*

$$\sum_k x_k y_k \quad \le \quad \left(\sum_k x_k^p\right)^{1/p} \left(\sum_k y_k^q\right)^{1/q}.$$

Let the pure data parallel running time be t_D, the optimal switched execution time be t_S, and the optimal mixed parallel running time be t_M ($t_M \le t_S \le t_D$). We will show that, among all possible batches, a balanced batch poses the worst instance for data parallelism and thus provides the greatest potential for improvement through mixed parallelism. Of course, certain quantities have to remain invariant over the space of maximization. Two invariants are possible:

1. The total work $\sum_i f(N_i) = F$, a constant.
2. The total size $\sum_i N_i = N$, a constant.

We will consider both variations.

Lemma 3.6 *With t_D, t_M, and f defined as above,*

$$\frac{e_M}{e_D} \quad = \quad \frac{t_D}{t_M} \quad \le \quad \frac{1}{e_\infty} + \frac{\sigma P}{e_\infty} \left(\frac{\sum_k f(N_k)/N_k}{\sum_k f(N_k)}\right).$$

Proof. Immediate, using $t_M \ge \frac{1}{P} \sum_k f(N_k)$. ■

The remaining exercise it to bound from above the parenthesized term in the above RHS.

Theorem 3.7 *Subject to either invariant,*

$$\frac{e_M}{e_D} \quad \le \quad \frac{1}{e_\infty} \left(1 + \frac{\sigma P L}{N}\right).$$

Proof. Here we will do a continuous analysis, assuming N_i's are real, rather than integers. Also assume $N_i > 1$ to avoid problems near zero.

For constant F, the quantity in parentheses is maximized when all $f(N_i) = F/L$. This follows since $x/f^{-1}(x) = x^{1-1/a}$ which is convex for $a > 1$.

For constant N, using $p = a/(a-1)$ and $q = a$ in Theorem 3.5, we obtain

$$\sum_k N_k^{a-1} \cdot 1 \quad \le \quad \left(\sum_k N_k^a\right)^{1-\frac{1}{a}} \cdot L^{1/a}.$$

Since $\sum_k N_k^a \ge L(N/L)^a = N^a/L^{a-1}$, we have

$$\frac{\sum_k N_k^{a-1}}{\sum_k N_k^a} \quad \le \quad \frac{L^{1/a}}{\left(\sum_k N_k^a\right)^{1/a}} \quad \le \quad \frac{L}{N}.$$

This value is achieved when all N_k are set to N/L. ■

It can be verified that the claim also holds for functions of the form $f(x) = x \log x$, etc., so the claim is quite broadly applicable.

3.3 Switching heuristic for unbalanced batch

In the previous section we bounded the maximum possible gain in efficiency of mixed over data parallelism. In this section we analyze an intuitive heuristic for a simpler execution model: execute some of the largest tasks in data parallel fashion, and pack the remaining small tasks into a task parallel phase.

The input instance is a set of independent tasks. Task i has size N_i and sequential running time $f_i = f(N_i)$ which increases with N_i. Assume all N_i are large enough that $(1/P + \sigma/N_i) < e_\infty$ (otherwise these small tasks would clearly be better off in the task parallel phase). The time to run task i in data parallel mode is $T_i = f(N_i)\,(1/P + \sigma/N_i)/e_\infty$, which increases with N_i. The problem is to decide for each whether to execute it in data parallel mode or task parallel mode.

Let $\mathsf{Pack}(P, S)$ be the makespan (length of schedule) generated by packing tasks from set S in task parallel mode into P processors. There are heuristics that return $\mathsf{Pack} \le (1+\epsilon)\,\mathsf{Pack}_{\mathrm{OPT}}$ for any given $\epsilon > 0$, within time that is polynomial in $|S|$ [22]. It is easy to see that $\mathsf{Pack}_{\mathrm{OPT}} \le \frac{1}{P}\sum_{s\in S} f_s + \max_{s\in S} f_s$. Consider the following heuristic.

> Prefix-Suffix
> Sort tasks in decreasing order: $N_1 > N_2 > \cdots > N_L$.
> For $1 \le i \le L+1$
> Define $p[i] = \sum_{1\le j<i} T_j$ ($p[1] = 0$).
> Define $s[i] = \mathsf{Pack}(P, \{i, \ldots, L\})$ ($s[L+1] = 0$).
> Pick $1 \le i^* \le L+1$ such that $p[i^*] + s[i^*]$ is minimal.
> Run tasks $1, \ldots, i^* - 1$ in data parallel mode.
> Run tasks $i^*, \ldots, L$ in task parallel mode.

Suppose tasks $i, \ldots, L$ have to be scheduled using switched parallelism, given the constraint that the largest task i has to be in the task parallel phase. Let $s^*[i]$ be the constrained optimal switched makespan.

Lemma 3.8 $\mathsf{Pack}_{\mathrm{OPT}}(P, \{i, \ldots, L\}) \le 2\, s^*[i]$.

Proof. Because $s^*[i] \ge \max\{(f_i + \cdots + f_L)/P, f_i\}$. ■

Theorem 3.9 *For any given $\epsilon > 0$, algorithm* Prefix-Suffix *can, in polynomial time, produce a schedule of length at most $2\,(1+\epsilon)$ times* OPT, *the optimal makespan.*

Proof. Given an optimal schedule, we can locate the largest task ℓ executed in task parallel mode. Prefix-Suffix produces a schedule of length $p[i^*]+s[i^*] \le p[\ell]+s[\ell] = p[\ell]+\mathsf{Pack}(P, \{\ell, \ldots, L\}) \le p[\ell]+(1+\epsilon)\,\mathsf{Pack}_{\mathrm{OPT}}(P, \{\ell, \ldots, L\}) \le p[\ell] + 2\,(1+\epsilon)\,s^*[\ell] \le 2\,(1+\epsilon)\,(p[\ell]+s^*[\ell]) = 2\,(1+\epsilon)\,\mathrm{OPT}$, since $\mathrm{OPT} = p[\ell] + s^*[\ell]$. ■

4 Task graphs

In this section we evaluate the benefits of mixed parallelism for task graphs where each vertex is a parallelizable task. As mentioned before, we can make the tightest predictions for balanced divide and conquer trees which do not show a "packing loss" because the tasks on each level are all identical. Later we provide a weaker performance bound for irregular task graphs.

The motivation to study divide and conquer problems arises out of the relatively small degree of task parallelism

available in applications. Static task graphs, such as those generated from control flow graphs by parallelizing compilers, have a fixed small degree of task parallelism. For example, the benchmarks in [20] have 4–7 fold effective task parallelism and the signal processing applications in [24] have a 2–5 fold task parallelism. The task parallelism in climate modeling applications is typically no more than 4–6. Divide and conquer is a natural parallel programming paradigm with large amounts of task parallelism, since the task graph (tree) is dynamically generated, and its size depends on the problem size (unlike in the above static examples). This can potentially supplement data parallelism with more generous amounts of task parallelism.

4.1 Balanced trees

We begin by motivating our choice of task graphs outlined in Section 2.2. We include some examples, and characterize these example as (N, a, c, d) trees. In addition to mixed and data parallelism, we study an intermediate form called *switched parallelism*, which is easier to implement than general mixed parallelism but has most of the benefits. Finally, we compute the running times of full divide and conquer trees using these three kinds of parallelism and apply the results to our examples.

4.1.1 Applications

Eigenvalue algorithms. Eigenvalue algorithms exhibit mixed parallelism. For example, a recent implementation of a dense nonsymmetric algorithm [3] proceeds by successively separating the matrix into two submatrices, the union of whose eigenvalues are the eigenvalues of the original matrix. The root node has size $N = n^2$. If the separation is perfect, each child is of size $\frac{n}{2} \times \frac{n}{2}$, or $N/4$. Performing this separation requires $O(N^{3/2})$ FLOPS. This successive separation process forms a binary tree with $c = 2$, $d = 4$, and $a = 3/2$. (We scale time so that the constant in $O(N^{3/2})$ becomes one.) For symmetric matrices, an algorithm similar in spirit is the beta-function technique of Bischof *et al* [5]. An eigenvalue algorithm of a different flavor, but still from the divide and conquer category, is Cuppen's method for symmetric tridiagonal matrices, where we can actually split the matrix exactly in half all the time [7, 21] (although the costs of the children are not so simple).

Sparse Cholesky. We consider the regular but important special case of the matrix arising from the 5-point Laplacian on a square grid, ordered using the nested dissection ordering [13]. In this case one may think of dividing the matrix into 4 independent subproblems, corresponding to dividing the square grid into 4 subsquares, each of half the perimeter. The work performed at a node which corresponds to an $n \times n$ grid is $O(n^3)$; most of this cost is a dense Cholesky of a small $n \times n$ submatrix corresponding to the nodes on the boundaries of the subsquares. Thus $N = n^2$, $a = 3/2$, $c = 4$ and $d = 4$. We will also see that the results go over to matrices with planar graphs.

4.1.2 Parallel Scheduling Strategies

We will consider the following three strategies:

Data parallelism. The tasks in the tree are executed sequentially, with the optimum number of processors (1 or P) used for each task.

Mixed parallelism. Level ℓ in the tree is treated as a batch of d^ℓ independent tasks each of size N/c^ℓ, and using the optimal scheduling strategy of lemma 3.1.

Switched parallelism. This is a limited kind of mixed parallelism, in which each task runs on 1 or P processors, the machine switching between task and data parallelism as needed. For balanced trees, we use data parallelism down to some level in the tree, and then switch to task parallelism. This switch will occur no later than level $\log_d P$, since at this level there will be a frontier of P identical tasks, one for each processor to work at unit efficiency. Thus, switched parallelism will not be as efficient as optimal mixed parallelism, but it is much simpler to implement, so if its efficiency is nearly as good, it is an attractive option. Switched parallelism is used, for example, by Bischof *et al* [5].

We will let t_1, t_D, t_M, t_S and t_T denote the running times for sequential execution, data parallelism, mixed parallelism, switched parallelism and task parallelism, respectively, and $e_1 = 1$, e_D, e_M, e_S and e_T denote the corresponding efficiencies.

How to schedule switched parallelism. We may apply Lemma 3.1 to choose the optimal level ℓ_S at which to switch from data to task parallelism:

$$\ell_S \;=\; \min\left\{ \ell :\; e_\infty \;\le\; \frac{d^\ell}{P} + \frac{\sigma (cd)^\ell}{N} \right\}. \qquad (4)$$

While we cannot write down a closed form expression for ℓ_S, it is easy to evaluate numerically, as well as to examine the limiting cases of $N/P \gg \sigma$ and $N/P \ll \sigma$. When $N/P \gg \sigma$, the first term in the RHS dominates and so we switch when $\ell > \log_d(Pe_\infty)$, i.e. when the number of tasks d^ℓ at level ℓ exceeds the maximum possible speedup Pe_∞. When $N/P \ll \sigma$, the second term in the RHS dominates and so we switch when $\ell > \log_{cd}(Ne_\infty/\sigma)$.

How to schedule mixed parallelism. We may again apply Lemma 3.1 to choose the optimal processor allocation for each level of the tree. Analogous to switched parallelism, there is a level ℓ_M at which one switches from mixed to task parallelism:

$$\ell_M \;=\; \min\left\{ \ell :\; e_\infty \le \frac{d^\ell}{P} + \frac{\sigma c^\ell}{N} \right\}.$$

As before, if $N/P \gg \sigma$ then the first term in the RHS dominates and so we switch when $\ell > \log_d(Pe_\infty)$. If $N/P \ll \sigma$, we switch when $\ell > \log_c(Ne_\infty/\sigma)$. Notice that, everything else being fixed, $\ell_S \le \ell_M$.

4.1.3 Comparing all the alternatives

We can collect all the preceding analyses into equations for the execution times for the sequential program (t_1), and parallel programs with pure data parallel (t_D), switched (t_S) and mixed (t_M) strategies.

$$\begin{aligned}
t_1 &= \textstyle\sum_{0 \le \ell \le \log_c N} d^\ell f\left(\frac{N}{c^\ell}\right) \\
t_D &= \textstyle\sum_{0 \le \ell \le \log_c N} d^\ell f\left(\frac{N}{c^\ell}\right) \min\left\{1,\; \frac{1}{e_\infty}\left(\frac{1}{P} + \frac{\sigma c^\ell}{N}\right)\right\} \\
t_S &= \textstyle\sum_{0 \le \ell \le \log_d P} f\left(\frac{N}{c^\ell}\right) \min\left\{1,\; \frac{d^\ell}{e_\infty}\left(\frac{1}{P} + \frac{\sigma c^\ell}{N}\right)\right\}
\end{aligned}$$

$$
\begin{aligned}
& + \quad \sum_{\log_d P < \ell \le \log_c N} \frac{d^\ell}{P} f\left(\frac{N}{c^\ell}\right) \\
t_M \quad = \quad & \sum_{0 \le \ell \le \log_d P} f\left(\frac{N}{c^\ell}\right) \min\left\{1,\ \frac{1}{e_\infty}\left(\frac{d^\ell}{P} + \frac{\sigma c^\ell}{N}\right)\right\} \\
& + \quad \sum_{\log_d P < \ell \le \log_c N} \frac{d^\ell}{P} f\left(\frac{N}{c^\ell}\right)
\end{aligned}
$$

Notice how the essential difference between these expressions is made by the position of the d^ℓ term in the first sum. Even after assuming the form N^a for $f(N)$, these have cumbersome closed forms, but they can be evaluated using few lines of MATLAB code, which we do for the simulation studies that follow. They can also be numerically solved by a runtime scheduler very quickly, compared to the bulk of the tasks.

Our next step is to estimate the performance difference between mixed and switched parallelism. We show that for trees large enough at the root, switched parallelism does very well, leaving little room for improvement. The basic intuition is that high up in the tree, a vertex has problem size large enough that data parallelism is efficient, while lower down, there are many problems to support task parallelism.

Lemma 4.1 *For (N, a, c, d) divide and conquer trees in the asymptotic model, the relative benefit of mixed over switched parallelism is*

$$
\frac{t_S - t_M}{t_S} \quad < \quad \frac{\sigma P}{e_\infty N} \sum_{0 \le \ell < \ell_M} \left(\frac{d}{c^{a-1}}\right)^\ell .
$$

4.1.4 Simulated performance

The space of programs, machines, and problem sizes is too large to examine completely; therefore we take some slices through this space that give insight into the benefits of mixed parallelism for typical current architectures and our suite of scientific programs. The following graphs are shown.

1. We fix $P = 128$, $e_\infty = 1$, $a = 3/2$, and $c = d = 4$ (as in sparse Cholesky), and plot e_D/e_M and e_S/e_M against σ (log-scale) in figure 3. The memory per node is assumed to be 64 MBytes, and the four plots correspond to problem sizes that fill 25, 50, 75 and 100% of the memory. For typical values of σ for various machines see table 1.
2. For the same sparse Cholesky problem, we consider four machines. In each case, the x-axis is P. N is such that the memory is completely filled. We plot e_M, e_S, and e_D against $\lg P$ in figure 4.
3. The setting is as above, except that typical values of P are chosen for each machine and the x-axis is $n = N^{1/2}$ (log-scale). See figure 5.
4. The setting is as in item (3), but the problem is the sign function program ($c = 4, d = 2$). See figure 6. For each task, as a reasonable estimate, there are 15 LU's, 15 BS's and 8 MM's. For this compound data parallel task, estimates of σ for various machines are shown in column SF of table 1 ($e_\infty = 1$). If Cuppen's eigenvalue algorithm is used, and the effect of "deflation" is small [7], the task tree has the same parameters as the sign function example above, although σ is different.

Comments. From table 1, typical values of σ are all in the 10^2 to 10^6 range. Throughout this range, switched parallelism appears to make up for much of the deficit in data parallel performance. The non-monotonicity in figure 3 occurs because after σ becomes absurdly large ($> 10^6$), parallelism is no longer effective. In general, for fine-grain MPP-class machines, mixed parallelism has little marginal benefit, while for more coarse-grain networks of workstations, switched parallelism is adequate. The choice of strategy is dictated not only by σ, N and P, but also the size reduction factor c and branching degree d. This is seen in figures 5 and 6: mixed parallelism gives less marginal benefit over switched or data for small values of d and large values of c (and vice versa). Figures 5 and 6 exhibit troughs because at the lower end of problem sizes, absolute efficiency of all the strategies are very small but close to each other.

4.2 Irregular graphs

In this paper we have mostly addressed regular problems. For batches, we established that a balanced batch makes data parallelism perform worst. Suppose we want to estimate the performance gains from mixed parallelism applied to an arbitrary task graph G. As mentioned before, it depends on scalability of the data parallel vertices as well as the amount of task parallelism. However, as we shall see, irrespective of the task graph, the maximum possible gains from mixed parallelism is related only to the number of vertices L, and either the sum N of the problem sizes at the vertices, or the total computation cost F of the graph.

Theorem 4.2 *The maximum benefit from mixed parallelism for a task graph G with L vertices such that the sum of problem sizes is N can be bounded as*

$$
\frac{e_M}{e_S} \quad \le \quad \frac{e_M}{e_D} \quad \le \quad \frac{1}{e_\infty}\left(1 + \frac{\sigma P L}{N}\right),
$$

in the asymptotic model using P processors.

PROOF. Given an irregular graph G with some (unknown) value of t_D/t_M, we will transform G into a batch G' such that the running time ratio $t'_D/t'_M \ge t_D/t_M$. To do this simply delete all dependency edges from G: clearly $t'_D = t_D$ and $t'_M \le t_M$. Next invoke Theorem 3.7, picking all $N_k = N/L$ or $f^{-1}(F)/L$, depending on whether the invariant is constant N or constant F. Thus over the space of all possible task graphs, the best one for mixed parallelism is a balanced independent batch. ■

We recall that we have given the mixed strategy the benefits of ideal efficiency and no precedence edges. Thus the above bound is overly optimistic. Even so, it is useful for deriving heuristic bounds to performance in some irregular graphs. E.g., Gilbert and Tarjan study nested dissection algorithms to solve sparse systems on planar graphs [14], where a problem of size N is divided into $d = 2$ subproblems, where each part is no bigger than $2N/3$. No matter what strategy we use in the upper levels, we only need to go down to roughly $\ell(\epsilon) = \frac{1}{\lg(3/2)}(1 - \frac{\log \epsilon}{\log P}) \approx 1.71(1 - \frac{\log \epsilon}{\log P})$ levels before the largest leaf is of size at most $\epsilon n/P$. At this point task packing is at most $(1 + \epsilon)$ times optimal. Given that there is not much need to go below this level, $L \le P^{1.71}$, so the maximum benefit is bounded above roughly by $t_S/t_M \le t_D/t_M \le 1 + \sigma P^{2.72}/N$.

5 Experiments

Our analysis in this paper was in part motivated by plans to implement a parallel divide and conquer program for finding all eigenvalues of a dense non-symmetric matrix using the

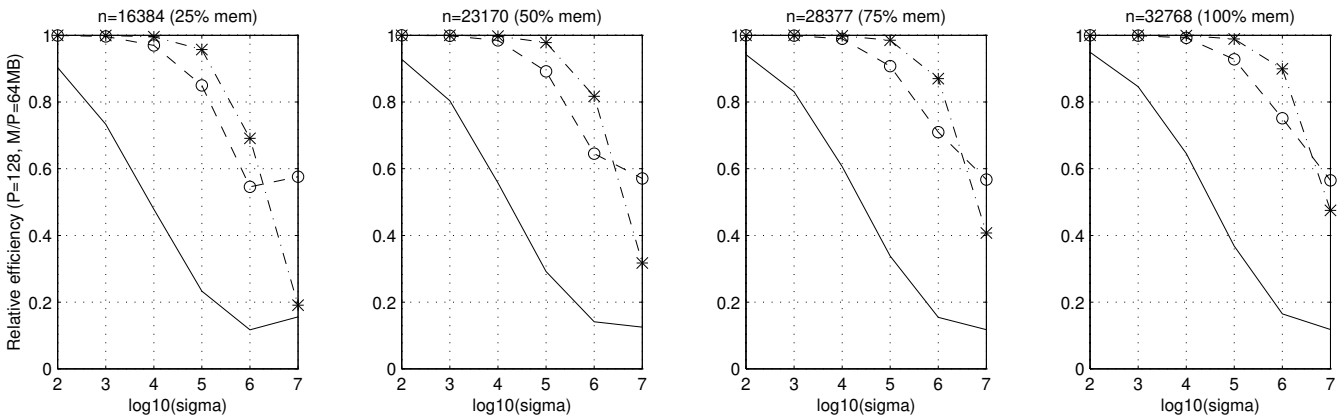

Figure 3: Variation of efficiency with σ in sparse Cholesky. The line with stars is absolute efficiency of mixed parallelism (e_M). The dashed line with circles is the relative efficiency e_S/e_M of switched parallelism. The solid line is the relative efficiency e_D/e_M of data parallelism.

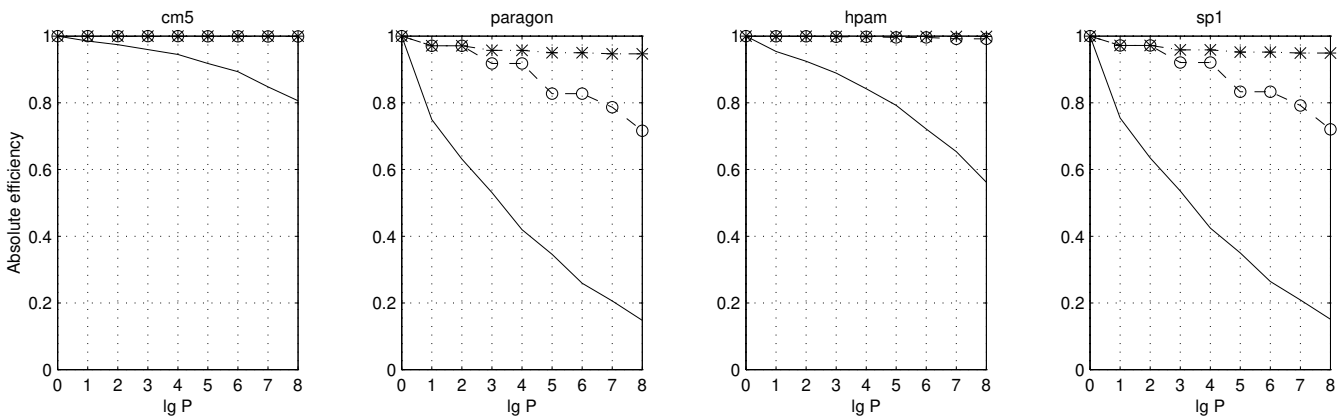

Figure 4: Scalability of sparse Cholesky on four machines using mixed, switched, and data parallelism. e_M (starred), e_S/e_M (circled), and e_D/e_M (solid) are plotted against $\lg P$, always choosing N to fill all memory.

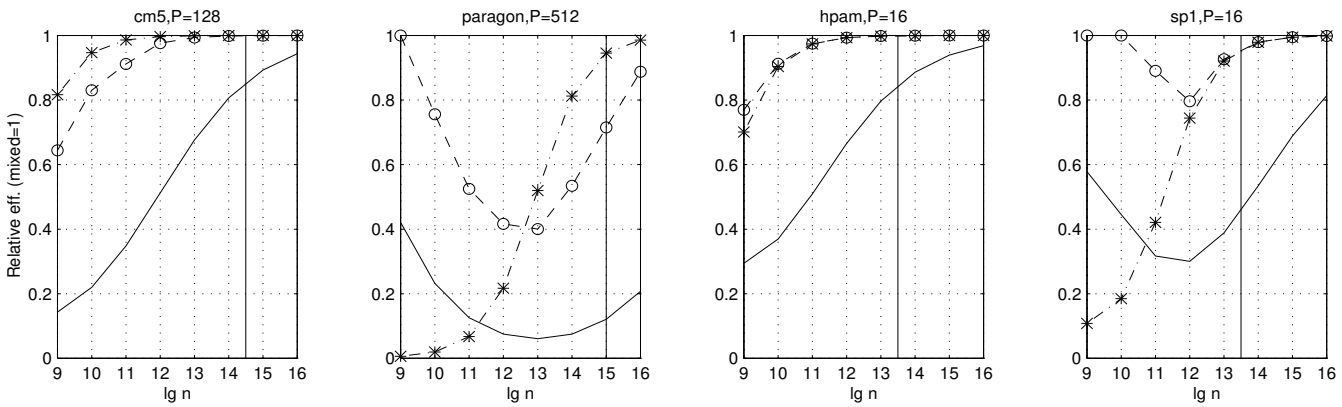

Figure 5: The setting is as in figure 4, except that typical values of P are chosen for each machine and the x-axis is $n = N^{1/2}$ (log-scale). Maximum size limit is shown by the vertical bar, where memory per node is from table 1. The line with stars is absolute efficiency of mixed parallelism (e_M). The dashed line with circles is the relative efficiency e_S/e_M of switched parallelism. The solid line is the relative efficiency e_D/e_M of data parallelism.

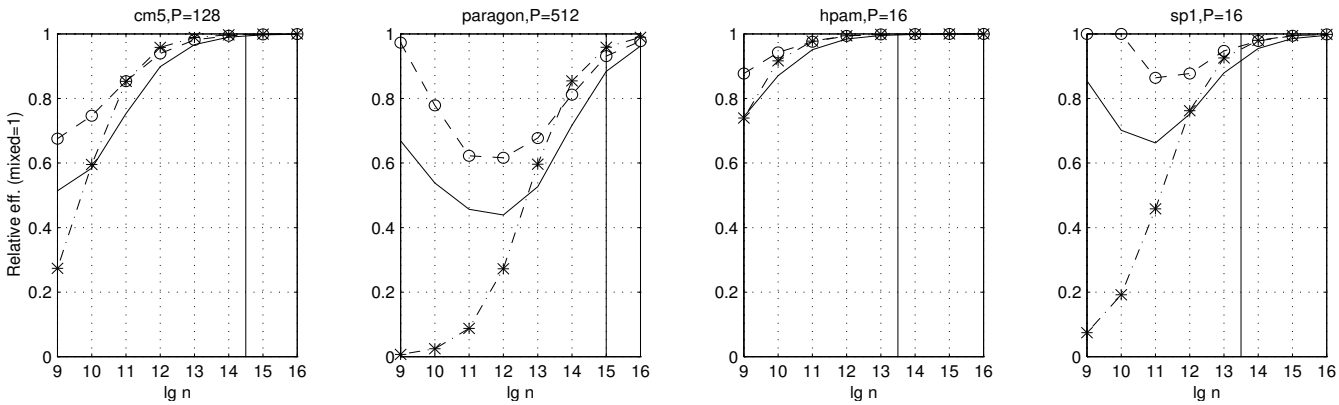

Figure 6: The setting is as in figure 5, except that the problem is changed to sign function, with parameters $c = 4$, $d = 2$, and a different value of σ.

sign function algorithm described earlier. The task tree in this program is generated on-line as "divide" tasks are executed. Unlike in our simplified analysis, the scheduler has to deal with an irregular frontier of ready tasks at any time. Based on our analyses, we implemented a switching scheduler along the lines of the irregular batch heuristic Prefix-Suffix described earlier.

```
Prefix-Suffix-Ready
  Initialize ready list with root task.
  While ready list is non-empty
    Call Prefix-Suffix.
    If i* = 1 (task parallel for all tasks)
      Break out of while loop.
    Remove a task from ready list.
    Run a data parallel divide step.
    Add children tasks to ready list.
  Pack the remaining tasks into P processors.
  Solve (conquer) the leaves independently on each processor.
```

Our prototype is coded on top of the ScaLAPACK scientific library running on a variety of machines. Here we report preliminary results on the Intel Paragon (Table 2). The input matrix is filled with random double precision numbers uniformly distributed in $(-1, 1)$. There is some anomaly (the $*$) owing to discrete switching effect. The observed gain from switched parallelism is somewhat larger than our predictions because the leaves use a efficient sequential algorithm different from the internal nodes, unlike the assumption in our model. We should be able to obtain better absolute performance by reducing some buffer copying overheads currently necessitated by temporary restrictions in ScaLAPACK.

6 Possible extensions

We made several simplifying assumptions in our analysis. For the problems analyzed in detail, they were reasonable. To expand the applicability of our approach, some extensions can be made. The most important extension is to tighten the bounds for irregular trees and DAGs. Other minor concerns are in the modeling choices. As an example, in Strassen's matrix multiplication, $f(N) = O(N)$, i.e., $a = 1$, so communication between tasks cannot be ignored. Other problems worthwhile exploring are sorting and FFT. In any case, the recipe for performance evaluation remains the same; only estimates of parameters and the efficiency profile equation may be tailored. Another extension could be to deal with non-homogeneous processor networks or non-homogeneous tasks, for example where one task is much better suited to one machine than another [19].

7 Conclusions

We have built a simple model to evaluate the utility of mixed data and task parallelism, with the goal of identifying how the communication cost and graph structure of the program, and network performance of the machine, affect the performance gain from using mixed parallelism. Our work provides the following.

- A simple general model for the efficiency of data parallelism that is validated against detailed performance models and empirical timing data.
- A simple formula for the benefits of mixed parallelism over data parallelism for a batch of equal-sized tasks.
- An upper bound on the benefit of mixed over data parallelism for tasks graphs of arbitrary shape and arbitrary task sizes, assuming all tasks have the same efficiency profile.
- A simple heuristic for scheduling an unbalanced batch of tasks, which is $2\,(1+\epsilon)$-optimal.
- Simple formulae for the efficiency of mixed, data, and switched parallel for a regular divide and conquer task tree.
- Examples to show that switched parallelism attains most of the benefit of mixed parallelism for a variety of interesting scientific problems.

These results have two implications. First, research results on mixed parallelism should report the performance gain over (at least) the space of parameters we have studied; the gain at a particular value of N, P, or σ may be misleading. Second, to avoid unnecessary coding complexity and/or runtime overhead, it is important to make rough estimates of performance gains before picking data, task, mixed, or switched parallelism.

Acknowledgements. Thanks to Jaeyoung Choi, Ren-Cang Li, Xiaoye Li, Ken Stanley, Mike Mitzenmacher, S. Muthukrishnan, and the anonymous referees.

n	t_D(s)	t_S(s)
100	10.14	1.19
200	34.18	5.74
300	93.59	18.92
400	817	259
500	1211	366
600	1702	491
700	2205	912
800	2495	832*
900	3181	1031

Table 2: Relative performance of data and switched parallelism in the sign function program. $P = 24$.

References

[1] R. Arpaci, D. Culler, A. Krishnamurthy, S. Steinberg, and K. Yelick. Empirical evaluation of the CRAY-T3D: A compiler perspective. In *International Symposium on Computer Architecture*. ACM SIGARCH, 1995.

[2] S. B. Baden. Programming abstractions for dynamically partitioning and coordinating localized scientific calculations running on multiprocessors. *SIAM Journal on Scientific and Statistical Computing*, 12(1):145–157, 1991.

[3] Z. Bai and J. Demmel. Design of a parallel nonsymmetric eigenroutine toolbox, Part I. In *Proceedings of the Sixth SIAM Conference on Parallel Proceesing for Scientific Computing*. SIAM, 1993.

[4] K. Belkhale and P. Banerjee. An approximate algorithm for the partitionable independent task scheduling problem. In *International Conference on Parallel Processing (ICPP)*. IEEE, August 1990. Full version in technical reports UILU-ENG-90-2253 and CRHC-90-15, University of Illinois, Urbana.

[5] C. Bischof, S. Huss-Lederman, X. Sun, A. Tsao, and T. Turnbull. Parallel performance of a symmetric eigensolver based on the invariant subspace decomposition approach. In *Scalable High Performance Computing Conference*, pages 32–39, Knoxville, TN, May 1994. IEEE.

[6] S. Chatterjee. Compiling data-parallel programs for efficient execution on shared-memory multiprocessors. Technical Report CMU-CS-91-189, CMU, Pittsburgh, PA 15213, October 1991.

[7] J. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.*, 36:177–195, 1981.

[8] J. Demmel, J. Dongarra, R. van de Geijn, and D. Walker. LAPACK for distributed memory machines: the next generation. In *Proceedings of the Sixth SIAM Conference on Parallel Proceesing for Scientific Computing*. SIAM, 1993.

[9] J. Demmel and K. Stanley. The performance of finding eigenvalues and eigenvectors of dense symmetric matrices on distributed memory computers. In *Proceedings of the Seventh SIAM Conference on Parallel Proceesing for Scientific Computing*. SIAM, 1994.

[10] F. Desprez, B. Tourancheau, and J. J. Dongarra. Performance complexity of *LU* factorization with efficient pipelining and overlap on a multiprocessor. Technical report, University of Tennessee, Knoxville, Feb 1994. (LAPACK Working Note #67).

[11] A. Feldmann, J. Sgall, and S.-H. Teng. Dynamic scheduling on parallel machines. In *Foundations of Computer Science (FOCS)*, pages 111–120, 1992.

[12] I. Foster, M. Xu, B. Avalani, and A. Chowdhary. A compilation system that integrates high performance Fortran and Fortran M. In *Scalable High Performance Computing Conference*, pages 293–300. IEEE, 1994.

[13] A. George and J. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.

[14] J. Gilbert and R. Tarjan. The analysis of a nested dissection algorithm. *Numerische Mathematik*, 50:377–404, 1987.

[15] X. Li and H. Huang. On the concurrency of C++. In *Proceedings ICCI '93. Fifth International Conference on Computing and Information*, pages 215–19, Ontario, Canada, May 1993.

[16] J. W. H. Liu. The multifrontal method for sparse matrix solution: theory and practice. *SIAM Review*, 34(1):82–109, March 1992.

[17] W. Ludwig and P. Tiwari. Scheduling malleable and nonmalleable parallel tasks. In *Symposium on Discrete Algorithms (SODA)*, pages 167–176. ACM-SIAM, 1994.

[18] S. Luna. Implementing an efficient portable global memory layer on distributed memory multiprocessors. Technical Report UCB/CSD-94-810, University of California, Berkeley, CA 94720, May 1994.

[19] C. R. Mechoso, C.-C. Ma, J. Farrara, J. A. Spahr, and R. W. Moore. Parallelization and distribution of a coupled atmosphere-ocean general circulation model. *Monthly Weather Review*, 121(7):2062–2076, 1993.

[20] S. Ramaswamy, S. Sapatnekar, and P. Banerjee. A convex programming approach for exploiting data and functional parallelism on distributed memory multiprocessors. In *International Conference on Parallel Processing (ICPP)*. IEEE, 1994.

[21] J. Rutter. A serial implementation of Cuppen's divide and conquer algorithm for the symmetric eigenvalue problem. Mathematics Dept. Master's Thesis available by anonymous ftp to tr-ftp.cs.berkeley.edu, directory pub/tech-reports/cs/csd-94-799, file all.ps, University of California, 1994.

[22] D. B. Shmoys and D. S. Hochbaum. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1):144–162, January 1987.

[23] K. Stanley and J. Demmel. Modeling the performance of linear systems solvers on distributed memory multiprocessors. Technical report, University of California, Berkeley, CA 94720, 1994. In preparation.

[24] J. Subhlok, J. Stichnoth, D. O'Hallaron, and T. Gross. Exploiting task and data parallelism on a multicomputer. In *Principles and Practice of Parallel Programming (PPoPP)*, pages 13–22, San Diego, May 1993. ACM-SIGPLAN.

[25] J. Turek, J. L. Wolf, and P. S. Yu. Approximate algorithms for scheduling parallelizable tasks. In *Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 323–332, 1992.

[26] C.-P. Wen and K. Yelick. Parallel timing simulation on a distributed memory multiprocessor. In *International Conference on CAD*, Santa Clara, CA, November 1993. An earlier version appeared as UCB Technical Report CSD-93-723.

[27] R. C. Whaley. Basic linear algebra communication subprograms: Analysis and implementation across multiple parallel architectures. Technical Report LAPACK working note 73, University of Tennessee, Knoxville, June 1994.