Extended Parallelism Models For Optimization On Massively Parallel Computers

M.S. Eldred and B.D. Schimel

Sandia National Laboratories[‡]
Albuquerque, NM 87185

1. Abstract

Single-level parallel optimization approaches, those in which either the simulation code executes in parallel or the optimization algorithm invokes multiple simultaneous single-processor analyses, have been investigated previously and have been shown to be effective in reducing the time required to compute optimal solutions. However, these approaches have clear performance limitations that prevent effective scaling with the thousands of processors available in massively parallel supercomputers. In more recent work, a capability has been developed for multilevel parallelism in which multiple instances of multiprocessor simulations are coordinated simultaneously. This implementation employs a master-slave approach using the Message Passing Interface (MPI) within the DAKOTA software toolkit. Mathematical analysis on achieving peak efficiency in multilevel parallelism has shown that the most effective processor partitioning scheme is the one that limits the size of multiprocessor simulations in favor of concurrent execution of multiple simulations. That is, if both coarse-grained and fine-grained parallelism can be exploited, then preference should be given to the coarse-grained parallelism. This analysis was verified in multilevel parallel computational experiments on networks of workstations (NOWs) and on the Intel TeraFLOPS massively parallel supercomputer.

In current work, methods for exploiting additional coarse-grained parallelism in optimization are being investigated so that fine-grained efficiency losses can be further minimized. These activities are focusing on both algorithmic coarse-grained parallelism (multiple independent function evaluations) through the development of speculative gradient methods and concurrent iterator strategies, and on function evaluation coarse-grained parallelism (multiple separable simulations within a function evaluation) through the development of general partitioning and nested synchronization facilities. The net result is a total of four separate levels of parallelism which can minimize efficiency losses and achieve near linear scaling on massively parallel computers.

2. Keywords

parallel optimization, multilevel parallelism, massively parallel computing

3. Introduction

The opportunities for exploiting parallelism in optimization can be categorized into four main areas[1]:

- *Algorithmic coarse-grained parallelism*: This parallelism involves the concurrent execution of multiple independent function evaluations and can be exploited in gradient-based algorithms (e.g., finite difference gradient evaluations, speculative optimization), nongradient-based algorithms (e.g., genetic algorithms, parallel direct search), approximate methods (e.g., design of computer experiments for building response surfaces), and concurrent iterator strategies (e.g., branch and bound, island-model genetic algorithms).
- *Algorithmic fine-grained parallelism:* This involves computing the basic computational steps of an optimization algorithm (i.e., the internal linear algebra) in parallel. This is primarily of interest in large-scale optimization problems and simultaneous analysis and design (SAND).
- *Function evaluation coarse-grained parallelism:* This involves concurrent computation of separable parts of a single function evaluation. This parallelism can be exploited when the evaluation of objective and constraint functions requires multiple independent simulations, e.g. multiple load cases, multiple uncoupled disciplinary analyses, etc.
- *Function evaluation fine-grained parallelism:* This involves parallelization of the solution steps within a single analysis code. Sandia-developed MP analysis codes include PRONTO, MPSalsa, COYOTE, ALEGRA, CTH, and many others.

By definition, coarse-grained parallelism requires very little inter-processor communication and is therefore essentially “free,” meaning that there is little loss in parallel efficiency due to communication as the number of processors increases. However, it is often the case in optimization that there are not enough separable computations on each cycle to utilize the thousands of processors available on MP machines. This limitation was illustrated in reference [2], in which a parallel coordinate pattern search optimization exploiting only coarse-grained algorithmic parallelism was shown to have a maximum parallel speedup of six. Since the coordinate pattern search algorithm has $2n$ independent evaluations per cycle, the implementation could only utilize at most six processors for a problem size of n=3 design variables.

Fine-grained parallelism, on the other hand, involves much more communication among processors and care must be taken to avoid the case of inefficient machine utilization in which the communication demands among processors outstrip the amount of actual computational work to be performed. This limitation was illustrated in reference [1] for a representative MPSalsa simulation in which it is shown that, while simulation run time does monotonically decrease with increasing number of processors, the relative parallel efficiency of the computation for fixed model size is decreasing rapidly (from $\hat{E}$ = 0.87 at 64 processors to $\hat{E}$ = 0.39

[‡]Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000. This paper is declared a work of the U.S. Government and is not subject to copyright protection in the United States

at 512 processors). This is due to the fact that the total amount of computation is approximately fixed, whereas the communication demands are increasing rapidly with increasing numbers of processors. Therefore, there is an effective limit on the number of processors that can be employed for fine-grained parallel simulation of a particular model size, and only for extreme model sizes can thousands of processors be efficiently utilized in optimization exploiting fine-grained parallelism alone.

These limitations point us to the exploitation of multiple levels of parallelism, in particular the combination of coarse-grained and fine-grained approaches. The question arises, then, if multiple types of parallelism can be exploited, how should the amount of parallelism at each level be selected so as to maximize the parallel efficiency of the study? Reference [1] answers this question in showing that the relative parallel efficiency $\hat{E}$ of a multilevel parallel optimizer on p processors, as a function of the size p' of multiprocessor simulations within p, is:

$$\hat{E}(p') \cong \frac{C}{p'\bar{T}(p')} \text{ for some } C > 0 \qquad (1)$$

where $\bar{T}(p')$ is the time required to run a multiprocessor simulation. For sublinear analysis speedup, the denominator is monotonically increasing with p' and $\hat{E}(p')$ is maximized at $p' = p'_{min}$. Therefore, the size of fine-grained parallel simulations is to be minimized by exploiting as much coarse-grained parallelism as possible. Important practical considerations and caveats include:

- Time spent in serial operations (T_{serial}) and in master-slave optimization communication (T_{comm}) is assumed to be negligible in comparison to $\bar{T}(p')$ (see [1] for derivation). The effect of nonnegligible T_{serial} and T_{comm} will be quantified in future work (with reference to Amdahl's law).
- Maximizing parallel efficiency and minimizing run time are not equivalent. Sometimes efficiency must be sacrificed for speed ($p' > p'_{min}$) in the case where additional processors are available. The analysis still provides useful guidance in this case.
- Transitory superlinear speedups can be caused by cache effects (as processor count increases, model partitions decrease in size and, at some point fit, may fit entirely in cache memory). In this case, maximum $\hat{E}(p')$ occurs at $\frac{d\hat{E}}{dp'} = 0$.
- For "heroic" scale problems, p'_{min} can stretch the limits of processor availability. In this case, coarse-grained parallelism cannot be used and the most effective usage must be made of the fine-grained parallelism (parallel SAND approaches [3],[4] appear very promising).

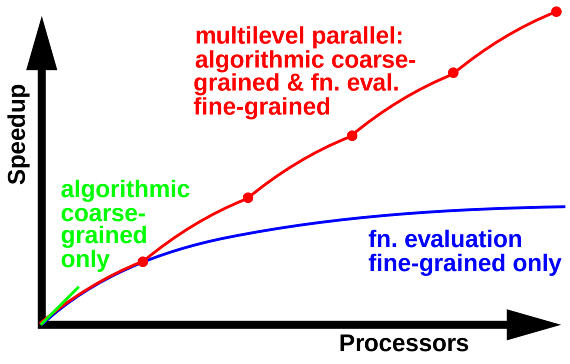

Figure 1. Scalability for fixed model size

Reference [1] also provides computational evidence for this conclusion by demonstrating minimum run time for $p' = p'_{min}$ on clusters of workstations and on the Intel TeraFLOPS MP computer.

The effect on scalability can be visualized as shown in Figure 1 in which the limitations of single-level parallelism are depicted along with the effect of combining the approaches into multilevel parallelism. By minimizing p', we move as far back on the fine-grained parallelism curve as possible, into the near-linear scaling range. And then we replicate this fine-grained parallel performance with multiple coarse-grained instances.

These insights have inspired current activities which are seeking to exploit additional coarse-grained parallelism within algorithms and function evaluations, as described in the following sections.

4. Algorithmic coarse-grained parallelism.

Parallel and speculative-parallel gradient-based methods

Quasi-Newton optimization algorithms often have a general structure consisting of a search direction computation, a value-based line search at one or more trial points in this direction for a sufficient decrease in the objective function, a gradient evaluation, an update to the Hessian approximation, and a convergence check. For these types of optimization algorithms, a load imbalance often occurs between the line search and gradient evaluations. For instance, for an optimization problem with n design variables using central finite difference gradients, $2n+1$ independent evaluations are needed to determine the function and approximate gradient values at a point, whereas only one independent evaluation exists when determining the function value during a value-based line search. Thus, a parallel implementation will have poor performance since most processors will be idle during the line search. A technique that addresses this imbalance, for the case of a value-based line search, is the speculative optimization approach [5]. By speculating that the current line search trial point will be successful, and thus the gradient information associated with the trial point will be needed directly afterwards, coarse grained parallelism can be introduced into the line search by computing the gradient information in parallel with the function value. Then the exact same number of independent evaluations exist during the gradient and line search phases and parallel implementations can be load balanced.

A common variation to the above optimization algorithm incorporates the use of a gradient-based line search. The accuracy of this type of line search is often enhanced through the use of a cubic approximation to the merit function, in place of the linear or quadratic approximations used in value-based line searches (note that these approximations are only used when the initial trial point is not a sufficient decrease). For the gradient-based line search, the gradient computation is no longer speculative since this information will always be requested, but the same load-balanced parallel implementation can be used. These parallel methods have been incorporated into DAKOTA [6] for several optimizer libraries capable of constrained nonlinear local optimization. No coding changes were necessary within the libraries, as it was sufficient to augment evaluation requests based on speculative logic within the wrapper classes.

Table 1 shows example parallel speedup and efficiency results for DAKOTA optimizations conducted on a network-connected workstation cluster. The OPT++ quasi-Newton vendor optimization algorithm [7] was used for this analysis to explore value-based load-imbalanced, value-based speculative, and gradient-based line searches. The results were generated using an algebraic nonlinear unconstrained Rosenbrock function [8], which was combined with a time delay of five seconds to simulate expensive function evaluations. The first two rows show results for the value-based line search in load imbalanced and speculative parallel modes, respectively. The value-based load-imbalanced approach requires one fewer processor than do the value-based speculative and gradient-based approaches because only a gradient computation is performed following the line search (the function value is already available). However the performance of the value-based load-imbalanced line search is much lower, as measured by speedup

$$S_p = \frac{T_s}{T_p} \tag{2}$$

where T_s and T_p are the wall clock times required for serial and parallel solutions, respectively, and efficiency

$$E(p) = \frac{S_p}{p} \tag{3}$$

where p is the total number of processors.

Table 1: DAKOTA/OPT++ speedup and efficiency results

Line Search Type	p	i_l	T_s (sec.)	T_p (sec.)	S_p	$E(p)$
value-based, load-imbalanced	4	45	986	435	2.27	0.57
value-based, speculative	5	45	986	239	4.12	0.83
gradient-based	5	40	1022	213	4.80	0.96

In parallel mode, the value-based speculative and gradient-based line search approaches offer much better performance. For this sample problem, the gradient-based line search offered the best overall performance, probably due to increased accuracy obtained by using a cubic approximation. This is evidenced by a faster parallel analysis time, T_p, resulting from fewer overall line search iterations, i_l. However, there are situations where the speculative line search can outperform the gradient-based line search. This can typically occur when the number of processors, p, is less than the number needed to compute all function and gradient-related function evaluations in one step, p_{fg}, where $p_{fg} = 2n + 1$ for central finite differencing. For these cases, when using a speculative value-based line search, any function and gradient-related function evaluations in excess of p can be terminated if the trial point is not accepted. Thus, if only a single processor is available, the series of function and gradient-related function evaluations observed for a value-based speculative line search reverts to what is observed for the standard value-based line search containing the load-imbalance (thus, T_s for load-imbalanced and speculative value-based line searches are equivalent). For a gradient-based line search, it will usually be necessary to compute all $p_{fg} > p$ evaluations at every trial point, which must be done in multiple steps and can often lead to longer overall solution times, T_p. Also, it should be cautioned that while it offers certain theoretical advantages in parallel mode over the value-based speculative line search, in practice the use of a gradient-based line search will not always show improved speedup performance, even if p_{fg} processors are available.

Concurrent iterator strategies

Additional coarse-grained parallelism on the algorithmic side can also be realized through the concurrent execution of multiple optimizations within a high-level strategy such as parallel branch and bound, multi-start local search, island-model genetic algorithms, etc. This requires development of an extended parallelism model in which we modularize the master-slave parallelism of an optimization scheduling function evaluations, such that multiple instances of this parallelism can be coordinated in a higher level master-strategy, slave-optimization approach. One can consider this design a multi-tiered master-slave: the strategy is a master to several slave optimization servers, each of which is a master to several slave function evaluation servers.

MPI [9] provides a convenient mechanism for modularizing parallelism through the use of communicators. A communicator defines the context of processors over which a message passing communication occurs. By providing mechanisms for subdividing existing communicators into new partitions and for sending messages between the new partitions, each level of parallelism can be managed with its own set of communicators independently from any other operations. And each of these new communicators can be further subdivided into additional partitions to nest multiple levels of parallelism, one within the other.

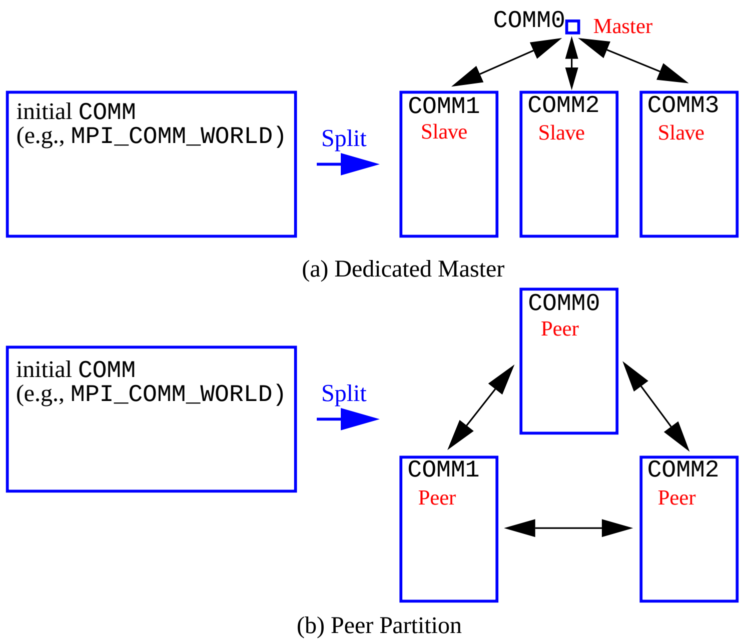

Figure 2. Communicator partitioning models

In DAKOTA, each tier within the nested parallelism hierarchy can use either of two processor partitioning models: a "dedicated master" partitioning in which a single processor is dedicated to scheduling operations and the remaining processors are split into server partitions, or a "peer partition" approach in which the loss of a processor to scheduling is avoided. These models are depicted in Figure 2. The peer partition is preferred since it avoids any idle time on a dedicated master; however it requires the use of either sophisticated mechanisms for distributed scheduling (to avoid work starved slaves) or a static partitioning of concurrent work for which scheduling is unnecessary.

Implementation of a concurrent iterator strategy within DAKOTA has focused on a parallel branch and bound strategy for optimizing nonlinear applications with mixed continuous-discrete variables. It employs the PICO branching engine [10] for scalable mixed integer programming combined with DAKOTA's nonlinear optimizers and multilevel parallelism facilities. Since PICO is capable of distributed scheduling, the peer partition model is used to manage the concurrent nonlinear programming optimizations within the strategy. To demonstrate this capability, the following mixed integer nonlinear programming problem is solved:

$$\text{minimize} \qquad f = \sum_{i=1}^{6} (x_i - 1.4)^4 \tag{4}$$

$$\text{subject to} \qquad g_1 = x_1^2 - \frac{x_2}{2} \le 0 \tag{5}$$

$$g_2 = x_2^2 - 0.5 \le 0 \tag{6}$$

$$-10.0 \le x_1, x_2, x_3, x_4 \le 10.0 \tag{7}$$

$$x_5, x_6 \in \{0, 1, 2, 3, 4\} \tag{8}$$

Since the evaluation of these functions is inexpensive, additional computational work (easily separable among processors for the multiprocessor simulation cases) is performed on each function evaluation so that the timings are more computation driven than communication driven. Preliminary results are provided in Table 2 where "iter$_{srv}$" denotes the number of concurrent iterator servers, "fev$_{srv}$" denotes the number of concurrent function evaluation servers used by each iterator, "ppe" denotes the number of pro-

Table 2: DAKOTA/PICO speedup and efficiency results

Levels of parallelism	p (iter$_{srv}$, fev$_{srv}$, ppe)	NLPs solved	T (sec.)	S_p	$E(p)$
0 (serial)	1 (1,1,1)	5	430.51	N/A	N/A
1	2 (2,1,1)	7	371.17	1.16	0.58
2	12 (2,5,1)	5	80.96	5.32	0.44
3	22 (2,5,2)	6	51.04	8.43	0.38
1	14 (1,13,1)	5	40.32	10.68	0.76
2	15 (1,7,2)	5	44.33	9.71	0.65

cessors per function evaluation, “NLPs solved” denotes the total number of nonlinear programming optimizations performed by the branch and bound strategy, T denotes the solution time for both serial (as marked) and parallel runs, and p, S_p, and $E(p)$ are as defined previously. When using the peer partition model in the strategy and the dedicated master model for each iterator, $p = \mathrm{iter_{srv}}(\mathrm{fev_{srv}ppe} + 1)$ for $\mathrm{fev_{srv}} > 1$ (parallel iterator) and $p = \mathrm{iter_{srv}fev_{srv}ppe}$ for $\mathrm{fev_{srv}} = 1$ (serial iterator).

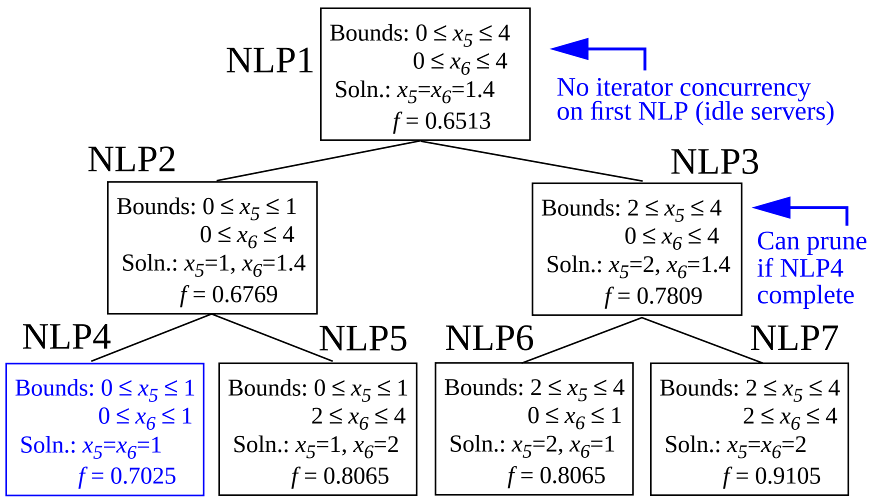
NLP1
Bounds: $0 \le x_5 \le 4$, $0 \le x_6 \le 4$
Soln.: $x_5=x_6=1.4$
$f = 0.6513$
No iterator concurrency on first NLP (idle servers)
NLP2
Bounds: $0 \le x_5 \le 1$, $0 \le x_6 \le 4$
Soln.: $x_5=1$, $x_6=1.4$
$f = 0.6769$
NLP3
Bounds: $2 \le x_5 \le 4$, $0 \le x_6 \le 4$
Soln.: $x_5=2$, $x_6=1.4$
$f = 0.7809$
Can prune if NLP4 complete
NLP4
Bounds: $0 \le x_5 \le 1$, $0 \le x_6 \le 1$
Soln.: $x_5=x_6=1$
$f = 0.7025$
NLP5
Bounds: $0 \le x_5 \le 1$, $2 \le x_6 \le 4$
Soln.: $x_5=1$, $x_6=2$
$f = 0.8065$
NLP6
Bounds: $2 \le x_5 \le 4$, $0 \le x_6 \le 1$
Soln.: $x_5=2$, $x_6=1$
$f = 0.8065$
NLP7
Bounds: $2 \le x_5 \le 4$, $2 \le x_6 \le 4$
Soln.: $x_5=x_6=2$
$f = 0.9105$

Figure 3. Branching for example problem

To fully understand the Table 2 comparisons, one must first consider the branching tree for this problem as shown in Figure 3, since the NLPs solved during the branch and bound process are dependent on parallelism and race conditions. For serial branch and bound, five NLPs are solved in the order numbered (NLP2 is sub-branched before NLP3 because its solution is better), and the solution to NLP4 allows pruning of the NLP3 branch, avoiding solution of NLP6 and NLP7 (note that this pruning is heuristic in the general nonlinear case since it would be possible for NLP6 or NLP7 to generate a better solution than NLP3 if the problem was multimodal, discontinuous, etc.). In parallel branch and bound with two concurrent iterators, the number of NLPs solved can be as high as seven since NLP2 and NLP3 are performed concurrently, NLP2 is followed by NLP4 and then NLP5 on one server, and NLP3 is followed by NLP6 and then NLP7 on the other server. To achieve scalable performance, the scheduling of NLPs is asynchronous and, if NLP4 on the first server is not completed prior to the initiation of NLP7 on the second server, then all seven NLPs will be performed. Thus, the solution of additional NLPs in the parallel case, as well as the fact that one server is idle during the solution of NLP1, yield poorer than desired parallel efficiencies for the $\mathrm{iter_{srv}}$=2 cases in Table 2 (rows 2, 3, and 4). Both of these effects would be less dominating for problems with more discrete variables resulting in a larger number of branches and NLP solutions. In the final rows of Table 2, serial branch and bound ($\mathrm{iter_{srv}}$=1) is combined with parallel optimization ($\mathrm{fev_{srv}}$=13 in row 5, $\mathrm{fev_{srv}}$=7 in row 6) and multiprocessor function evaluations (ppe = 2 in row 6) to yield more desirable speedup and parallel efficiencies. Thus, it can be concluded that implementing a parallel branch and bound which is scalable in the number of concurrent bounding operations is quite challenging and more work remains in tuning the use of PICO for solving nonlinear engineering applications with DAKOTA. Lastly, in all cases, the solution of seven or fewer NLPs compares well with the 25 solutions that would be required for simple enumeration of the discrete variable values.

5. Function evaluation coarse-grained parallelism

Within a single function evaluation, coarse-grained parallelism can be exploited if multiple separable simulations must be performed as part of evaluating the objective function(s) and constraints. This situation commonly arises in optimization when evaluation of the objectives and constraints involves simulations from multiple uncoupled disciplines (e.g., impact and fire survivability analyses), from multiple load cases or operating conditions (e.g., pull up and roll maneuvers), or from evaluating design robustness and insensitivity across parameter ranges.

In exploiting this type of coarse-grained parallelism, a third tier can be added to the two described above in which a function evaluation is a master to several multiprocessor slave analysis servers. DAKOTA is implementing this capability by providing an extended interface specification with an initial serial analysis portion for preprocessing (if required), followed by a set of concurrent analysis portions, followed by a final serial analysis portion for postprocessing (if required). The synchronization function for these analysis portions will be nested within the outer synchronization for concurrent function evaluations, such that a test for completion in the outer synchronize (on one of the concurrent function evaluations) will not complete successfully unless all of the analysis portions in the inner synchronize have completed.

With regards to the selection of a dedicated master or peer partition model, DAKOTA determines this at run time. If a sufficient allocation of processors is available to handle each of the concurrent analysis portions, then no scheduling is required ($N_{servers} = N_{jobs}$) and the more efficient peer partition model is used. If, however, not enough servers can be created to handle all of the concurrent multiprocessor analyses simultaneously ($N_{servers} < N_{jobs}$), then the dedicated master model is used so that each of the analyses can be self-scheduled through the servers without potential for work starvation.

6. Conclusions

Combining the extended parallelism concepts described in this paper result in a total of three nested tiers of master-slave control and four levels of parallelism. These extensions are depicted graphically in Figure 4. It is important to recognize that the effect is not additive, but rather multiplicative in nature. For example, if four concurrent optimizations can be run within a strategy, each

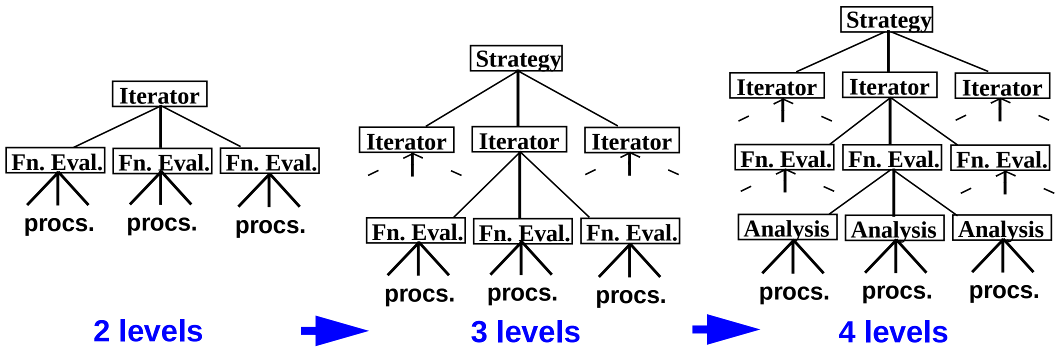

Figure 4. Extended parallelism

optimization having ten independent function evaluations on each cycle, and each function evaluation having three independent analyses, then the fine-grained parallelism available in the analysis can be augmented with 120-fold coarse-grained parallelism. The net effect of this additional coarse-grained parallelism is to further improve upon the multilevel parallel scaling of Figure 1 as shown in Figure 5.

In addition to continuing development of parallel branch and bound and concurrent analyses within a function evaluation, current work is addressing the practical realities of parallel optimization using shared massively parallel computing resources. Shared systems generally employ queuing software (e.g., NQS) which can have the detrimental effect of causing repeated delays (often hours, sometimes days) on *each* optimization cycle if they are used for queuing function evaluations. Alternative approaches of linking the simulation into the optimization code or ganging jobs together within special-purpose server scripts avoid the repeated delays, but suffer from other shortcomings (requiring modification to analysis codes or replication of already existing job management facilities in specialized scripting, respectively). Each of these approaches is currently under investigation for large-scale applications on the ASCI machines, the most fruitful of which will be selected for further refinement.

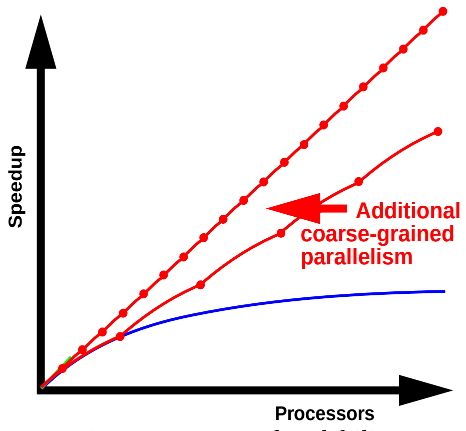

Figure 5. Improved scalability

7. References

[1]Eldred, M.S. and Hart, W.E., “Design And Implementation Of Multi-level Parallel Optimization On The Intel Teraflops,” paper AIAA-98-4707 in *Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, St. Louis, MO, Sept. 2-4, 1998, pp. 44-54.

[2]Eldred, M.S., Hart, W.E., Bohnhoff, W.J., Romero, V.J., Hutchinson, S.A., and Salinger, A.G., “Utilizing Object-Oriented Design to Build Advanced Optimization Strategies with Generic Implementation,” paper AIAA-96-4164 in *Proceedings of the 6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Bellevue, WA, Sept. 4-6, 1996, pp. 1568-82.

[3]Ghattas, O. and Orozco, C.E., “A parallel reduced Hessian SQP method for shape optimization,” *Multidisciplinary Design Optimization: State of the Art*, Natalia M. Alexandrov and M.Y. Hussaini, eds., SIAM, p.133-152, 1997.

[4]Biros, G. and Ghattas, O., “Parallel domain decomposition methods for optimal control of viscous incompressible flows,” to appear in *Proceedings of Parallel CFD ’99*, Williamsburg, VA, USA, May 23-26, 1999.

[5]Byrd, R.H., Schnabel, R.B., and Schultz, G.A., “Parallel quasi-Newton methods for unconstrained optimization,” *Math. Prog.*, 42(1988), pp. 273-306.

[6]Eldred, M.S, Bohnhoff, W.J., and Hart, W.E., “DAKOTA, An Object-Oriented Framework for Design Optimization, Parameter Estimation, Sensitivity Analysis, and Uncertainty Quantification,” Sandia Technical Report SAND99-XXXX, In preparation. Draft available from http://endo.sandia.gov/9234/sd_optim/Dakota_online.pdf.

[7]Meza, J. C., OPT++: An Object-Oriented Class Library for Nonlinear Optimization, Sandia Technical Report SAND94-8225, Sandia National Laboratories, Livermore, CA, 1995.

[8]Gill, P.E., Murray, W., and Wright, M.H., *Practical Optimization*, Academic Press, New York, 1981.

[9]Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J., *MPI: The Complete Reference*, MIT Press, Cambridge, MA, 1996.

[10]Eckstein, J., Hart, W.E., and Phillips, C.A., “Resource management in a parallel mixed integer programming package,” *Proceedings of the 1997 Intel Supercomputer Users Group Conference (http://www.cs.sandia.gov/ISUG97/program.html)*, Albuquerque, NM, June 11-13, 1997.