Scalability of Branch-and-Bound and Adaptive Integration

Rodger Zanny, Karlis Kaugars and Elise de Doncker
Department of Computer Science
Western Michigan University
Kalamazoo MI 49008

Abstract *We address the scalability of branch-and-bound and adaptive integration methods in the common framework of task refinement strategies. Not withstanding the different nature (discrete versus continuous) of the problems solved by these algorithms, we give an analogous treatment of scalability issues for these irregular problems, including notions of inherent limitations on the scalability of problem instances, and introduce a parallel scalability metric. Assessment and validation of the metric is based on detailed test results.*

Keywords: Branch-and-bound, adaptive integration, scalability, task refinement, parallel branch-and-bound.

1 Introduction

We address a class of algorithms which proceed by selecting and refining tasks. Using global selection criteria, the value of these methods is clear for tackling irregular problems, with problem hot spots given considerable and automatic attention through local refinement. As a result of inherently sequential problem behavior and global algorithm selection criteria, parallel performance is often affected by poor scalability. We explore similarities between adaptive partitioning methods for multivariate integration (cubature) and branch-and-bound strategies.

1.1 Terminology

An Adaptive Cubature (AC) algorithm attempts to approximate a given integral to within a requested accuracy, by successive refinement of the domain of integration. The subdivision structure of the domain allows for a tree representation of the examined subproblems. Each node in the *region subdivision tree* corresponds to a region, and its children nodes to the subregions obtained from the parent region within one subdivision step. At any stage of the algorithm, the leaf nodes correspond to the currently available (*active*) regions.

Specifically, a Global AC (GAC) method attempts to achieve the requested accuracy globally, over the entire active set of regions. The priority for subdivision is used to key a priority queue data structure. For example, the ParInt [2] GAC selects the next region based on its local estimated absolute integration error, and bisects the region perpendicularly to the direction where the integrand is estimated to vary the most. The subdivision tree is thus a binary tree. We use a heap-based priority queue to manage the active set.

Classically, a Local AC (LAC) method subdivides a region until a local acceptance criterion is reached (such as local estimated absolute error not exceeding the global absolute tolerance times the ratio of the subregion's volume to that of the original domain), upon which the subregion becomes inactive or is discarded. This subdivision procedure corresponds to a depth-first generation of the subdivision tree

and can be implemented using a stack.

Branch-and-bound (B&B) methods, which are used to search the state space of an optimization problem, proceed by partitioning the solution space. The entire space is represented at the root node of the corresponding B&B tree. The children at each node represent the subspaces obtained by *branching*, or subdividing the solution space of the parent node. Each iteration of a B&B procedure selects the next node from which to branch based on bounds estimated for the solution at each node. The node's children with their bounds are generated. Comparison of a child's bound with the current best solution (if one exists so far) allows for the discarding of inferior nodes. Non-discarded (live) nodes are added to the active set.

B&B strategies include LCB&B where the next node to branch from is selected as the one with Least Cost [3]. As in GAC, LCB&B requires the maintainance of a priority queue. As in LAC, the tree can be explored depth-first (using a stack), whereas breadth-first exploration corresponds to using a FIFO queue.

1.2 Irregular Problems / Scalability

GAC is generally considered superior to LAC methods especially for dealing with irregular problems, such as integrand singularities or peaks. The fact that GAC methods rely on a form of global priority queue, as well as updating of global data, presents a disadvantage with respect to parallelizing these algorithms, especially in a distributed environment. Maintaining a global heap is unsuited in view of communication expense [4]. Alternatively, multiple, distributed, local heap structures, which for irregular problems or on heterogeneous processors will grow and shrink at different rates, imply the need for load balancing, again incurring communication costs.

In GAC and LCB&B, processors can work on non-globally-optimal subproblems (given imperfect load balancing). Furthermore, there may not be enough "important" subproblems for all processors. As a consequence, there may be *work redundancy*, defined as the extra work (e.g., extra node expansions) done by the parallel algorithm, when compared with the sequential algorithm. Excessive work redundancy is termed *work anomaly* [5]. This is especially prevalent in GAC, as nodes are never eliminated. As a trade-off, processors are idled in the absence of any work to perform, as is the case in LAC and B&B strategies, where non-promising nodes are killed. Both work redundancy and idle time have an adverse and proportional effect on speedup and parallel efficiency.

Furthermore, given a basic problem solving method (such as GAC or B&B), the problems of redundancy and idle time is in some cases inherent to the specific problem being solved: no tweaking of the algorithm can avoid the anomalous behavior rooted in the problem instance.

Subsequently in this paper we will estimate the inherent non-scalability of certain irregular problem classes. Section 2 introduces a scalability metric which can be used for GAC and LCB&B. Section 3 gives results for test cases where the amount of inherent scalability in a problem is varied, resulting in corresponding changes both in terms of predicted behavior by the metric, and observed behavior using a synchronous global heap simulation for the active set. Section 4 gives conclusions and addresses future work.

2 Measures of Scalability

In [1], the number of *critical nodes* of a branch-and-bound problem instance is used as the scalability measure. A critical node is defined as a node in the search space whose bound function value is strictly less than the optimal solution. The set of critical nodes must be expanded by any sequential or parallel B&B algorithm to guarantee that the optimal solution is found. The number of critical nodes depends upon the problem, the problem instance, the bound function, and the branching method used.

If a problem has few critical nodes, there may not be enough available at any iteration of a parallel algorithm to keep a large number of workers busy performing useful work. For a given problem instance, a tight bound function will reduce the number of critical nodes, reducing the node expansions needed to solve the problem, but also reducing the scalability. A loose bound function will result in more critical nodes and better scalability, but more overall work [1].

Correspondingly, certain nodes within a GAC problem must be expanded to reach a solution. These regions at least include those of which the error estimate exceeds the total requested accuracy; these regions can be considered the *critical regions*. Without subdividing these regions, the requested accuracy cannot be reached. As in B&B, if the number of these critical regions is large, so is the amount of important work available to workers at any given iteration, and therefore the scalability improves. The number of critical regions depends upon the problem instance, the integration rule, and the subdivision strategy.

A related category of nodes is the set of *primary* nodes. These are the nodes that are expanded in a sequential execution of the B&B or GAC algorithm. Every critical node must clearly also be a primary node. The number of primary nodes is also a useful measure of the inherent scalability of a problem, as the sequential solving of a problem allows for all information (which would include global information in a parallel algorithm) to be available throughout the computation, and in the absence of speedup anomaly will result in a solution with a minimal number of node expansions. We have found this to be the case in GAC as well.

A weakness of using the number of these types of nodes as a measure of scalability is that it does not reflect the available work at any particular moment during the execution of the algorithm. A larger problem instance may have more critical nodes than a smaller problem instance, but at any given point in time it may not offer any additional amount of useful work to the workers, as the critical nodes are available only over the increased number of iterations it takes to solve the larger problem.

For both B&B and GAC, we propose as a useful measure of scalability the number of primary nodes that are on the algorithm's priority queue at any iteration, which we term the *current primary node count*. For B&B, this measure corresponds to the active set size during a sequential run, assuming that nodes on the heap that are dominated by a new best solution are removed from the heap as soon as the new solution is found

A related measure is of course the *current critical node count*, the current number of critical nodes on the priority queue.

Figure 1 shows the current primary region count, graphed across the iterations, for the integration of the function $f(x, y, z) = 1/(x + y + z)^2$, over the unit cube. The "ideal" line on the graph represents the maximum possible number of primary regions available at any iteration. In this ideally scalable case, every region placed on the heap is assumed to be a primary region; at the half-way point through the execution, the decreasing number of remaining iterations then begins to reduce the number of regions on the heap that could later be removed. If we define p_i to be the number of current primary regions at iteration i, then a useful measure is $c_i = \sum_{j=1}^{i} p_j$, the cumulative total of the primary node counts. Figure 2 plots the actual and maximum, or ideal, c_i values for the data from Figure 1.

3 Experimental Results

3.1 Adaptive Integration

To test the proposed scalability metric for adaptive integration we consider a problem which allows for varying the actual degree of scalability. The integrand function is of the form

$$f(\mathbf{x}) = \sum_{s=1}^{S} (\sum_{j=1}^{d} (x_j - \alpha_s)^2)^{\beta_s},$$

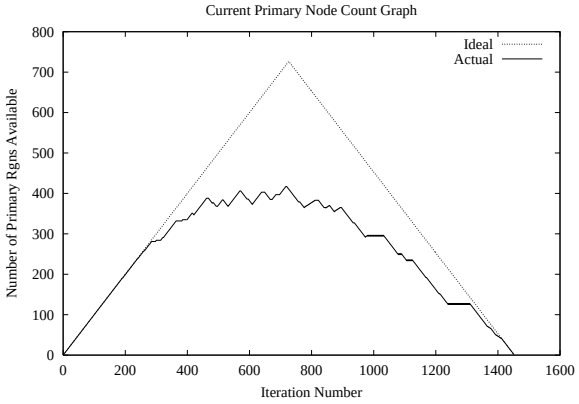

Figure 1: Plot of running primary region count

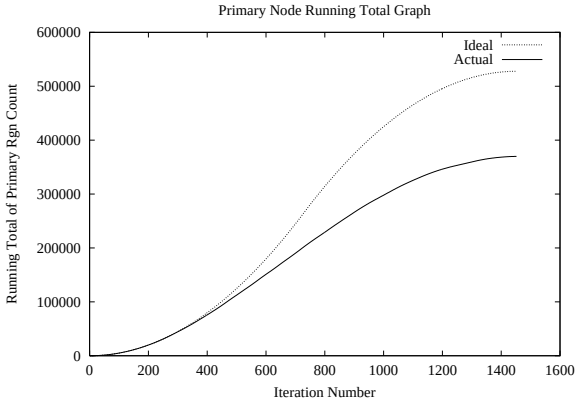

Figure 2: Plot of cumulative total of primary region count

with radial singularities (when $0 > \beta_s > -d$) at the points with coordinates $x_j = \alpha_s$, $j = 1, \ldots, d$ and $1 \leq s \leq S$, and integrated over the $d-$dimensional unit hypercube. The amount of available work thus increases with the number of singularities S.

As the purpose of the experiment is to test the ability to recognize the inherent degree of singularity as it affects scalability, separate from the parallel environment, we perform runs using a synchronous global heap simulation for p processors. At each iteration, all of the simulated workers synchronously remove from the heap and then subdivide p regions, evaluate the new subregions, and place the $2p$ new subregions back on the heap. The simulator allows for the effects of parallellization to be seen on the amount and pattern of work performed, but does not simulate timing.

Figures 3-8 show results of the experiment for an integral in $d = 3$ dimensions, $\beta_s = -1.25, 1 \leq s \leq S$, and $S = 1, \ldots, 20$. The singularities are restricted to a fairly small portion of the domain, $0 \leq \alpha_s \leq 0.25$, and are added consecutively.

Figure 3 plots the current primary region count versus the number of iterations for problems with an odd number of singularities S for $1 \leq S \leq 15$, and shows that the increase in scalability for increasing S is recognized effectively. Figure 5 shows the observed work redundancy (the ratio of the parallel region evaluations to the sequential region evaluations), as a function of the number of processors in the synchronous global heap simulation, which decreases for increasing S, matching our expectations. The work redundancy also decreases as the requested relative error is decreased for a fixed number of singularities, which is shown in Figure 6 for $S = 9$.

Figure 7 plots the observed work redundancy as a function of both the number of singularities (odd and even) and the requested accuracy. To allow for a 3-dimensional plot of primary region counts across multiple runs, Figure 8 plots the average number (over all iterations) of available primary regions as a function of the number of singularities (for S odd)

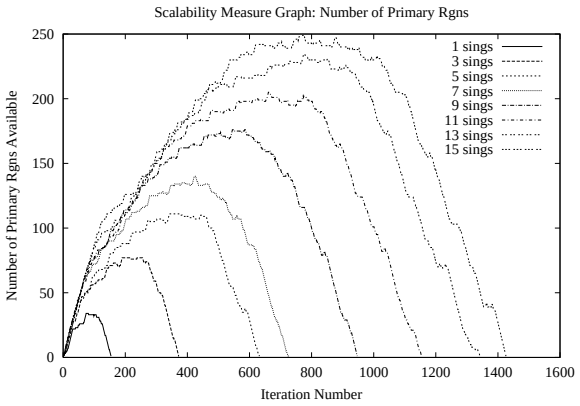

Figure 3: Current number of primary regions, for various numbers of singularities

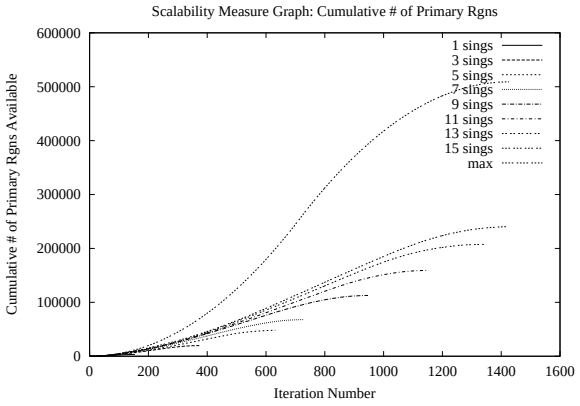

Figure 4: Cumulative number of primary regions, for various numbers of singularities

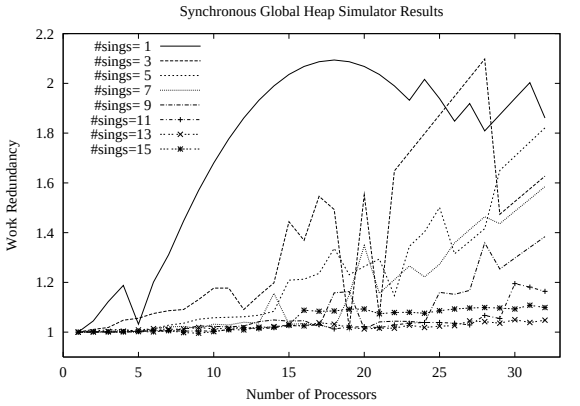

Figure 5: Plot of work redundancy vs. number of processors, for various numbers of singularities

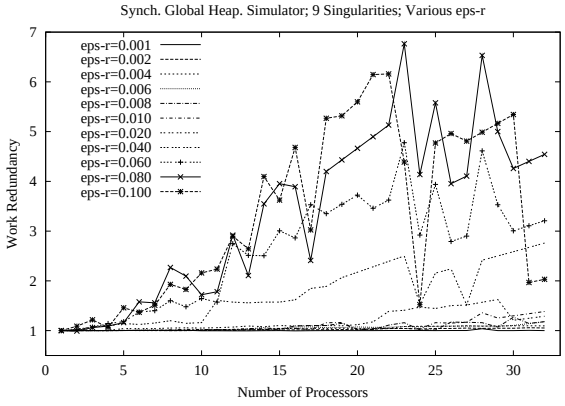

Figure 6: Plot of work redundancy vs. number of processors, for various requested accuracies

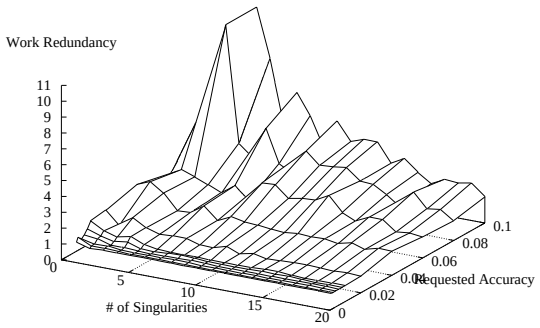

Figure 7: Work redundancy vs. number of singularities and requested accuracy

and the requested accuracy, conforming with the observed behavior depicted in Figure 7.

3.2 Branch-and-Bound

We consider instances of the traveling salesperson problem (TSP) for completely connected graphs of degree k, for $k = 3, \ldots, 8$. The state space is thus represented by a permutation tree where the root has $k-1$ children and the total number of nodes is $1+\sum_{j=0}^{k-2}\prod_{i=0}^{j}(k-i-1)$. For the experiment we use Least Cost Branch-and-Bound (LCB&B), with a naive bounding function equal to the cost of the path to each node. We report results of comparing our scalability

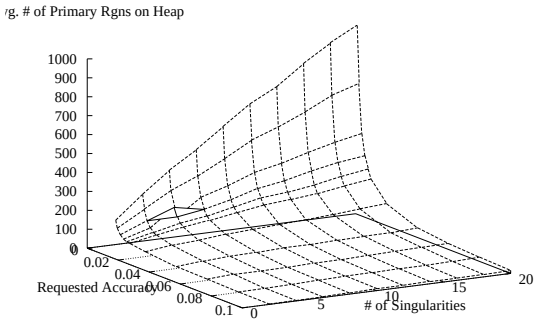

Figure 8: Number of Primary Regions vs. number of singularities and requested accuracy

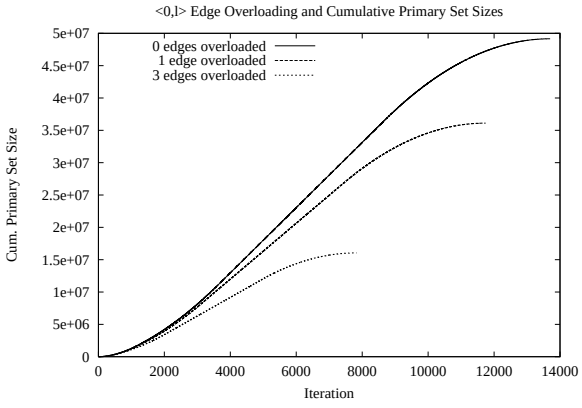

Figure 9: Cumulative primary set sizes vs. iteration with overloaded edges

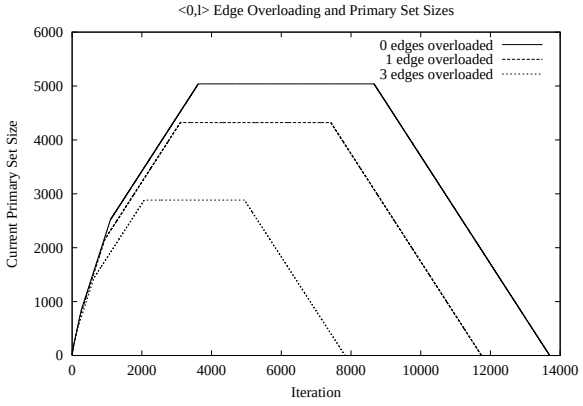

Figure 10: Current primary set sizes vs. iteration with overloaded edges

measure to work efficiency observed from the synchronous global heap simulator.

Starting from the instance with equal weights ($= 1$) on all edges, we change the weights by an amount greater than the length of the tour on an increasing number S of edges, giving rise to an increasing amount of pruning in the state space tree, and thereby decreasing the scalability. Note that a change to any edge affects subtrees in a similar manner. For example, assuming the path starts at vertex 1, applying the change to edge $<1, \ell>$ prunes the subtree at the child of the root corresponding to ℓ and causes similar effects where the changed edge appears within the other subtrees.

Figure 9 shows the cumulative primary node node set size for three fully connected graphs of degree 8. Setting all edges to a cost of 1 forces the algorithm to examine every tour in the graph and results in the maximum (over all iterations) primary node set size and iteration count. Setting a single $<1, \ell>$ to a value greater than the tour length reduces the maximum primary node set size by 719 nodes and shortens the number of iterations by 1957, while setting three such edges reduces the maximum by 2157 and shortens the iterations by 5871 steps. Note that for these regularly structured graphs, the primary node set is identical to the critical node set.

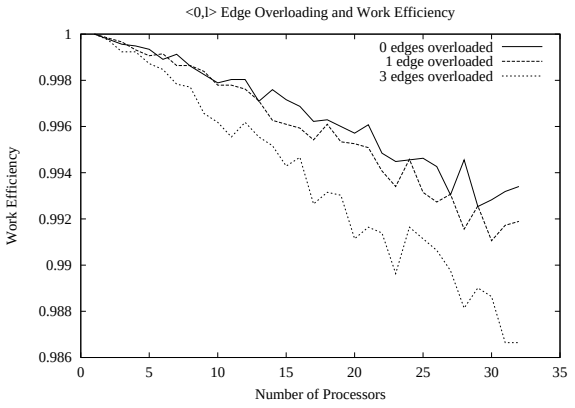

Figure 11: Work efficiency vs. number of processors with overloaded edges

We use the synchronous global heap simulator to measure the *work efficiency* for solving these TSP problems. Here, we define work efficiency as the ratio of the number of sequential node expansions to the number of parallel iterations, divided by the number of processes. This definition accounts for both redundant node expansions and workers idled due to a small active set size. To assess the proposed scalability metric we compare the observed work efficiency, as shown in Figure 11, with our scalability measure. As the scalability of the problem decreases (through the overloading of edge weights), work efficiency does undergo a corresponding decrease.

4 Conclusions and Future Work

The contributions of this paper include an analogous treatment of certain branch-and-bound and adaptive multivariate integration algorithms in the framework of task refinement. Concepts of branch-and-bound methods are introduced for adaptive integration, such as that of "critical nodes". Vice versa, ideas are extended to branch-and-bound strategies, such as that of "singular" problem behavior, and "adaptive" selection in LCB&B. Furthermore, a parallel performance metric is introduced to estimate the scalability of these methods for irregular problem classes. Test cases are constructed, providing elaborate test results for a synchronous global heap simulation.

Future work includes more extensive testing for different problem classes, for example to examine the effects of different bounding functions on scalability in branch-and-bound. Another aspect is that of speedup anomaly (where the parallelization "gets lucky" and searches a smaller space). Whereas the present paper focused more on the inherent sequential behavior of the problem, for which our approach using a global heap simulation is well-suited, we intend to also factor in effects from the parallel algorithm implementation and environment (which can only be done using actual parallel performance).

References

[1] Jens Clausen and J. L. Träff. Do inherently sequential branch-and-bound algorithms exist? *Parallel Processing Letters*, 4(1 & 2):3–13, 1994.

[2] E. de Doncker, A. Gupta, A. Genz, and R. Zanny. http://www.cs.wmich.edu/parint, PARINT Web Site.

[3] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1984.

[4] R. Zanny. Efficiency of distributed priority queues in parallel adaptive integration. Master's thesis, Western Michigan University, 1999.

[5] R. Zanny and E. de Doncker. Work anomaly in distributed adaptive partitioning algorithms. In *Proceedings of the High Performance Computing Symposium (HPC'00)*, pages 178–183, 2000.