Experiments with a Parallel Multi-Objective Evolutionary Algorithm for Scheduling

Matthew Brown
University of Southern California,
Los Angeles, CA, 90089
matthew.a.brown @ usc.edu

Mark D. Johnston
Jet Propulsion Laboratory/Calif. Inst. of Technology,
Pasadena, CA 91109
mark.d.johnston @ jpl.nasa.gov

Abstract

Evolutionary multi-objective algorithms have great potential for scheduling in those situations where tradeoffs among competing objectives represent a key requirement. One challenge, however, is runtime performance, as a consequence of evolving not just a single schedule, but an entire population, while attempting to sample the Pareto frontier as accurately and uniformly as possible. The growing availability of multi-core processors in end user workstations, and even laptops, has raised the question of the extent to which such hardware can be used to speed up evolutionary algorithms. In this paper we report on early experiments in parallelizing a Generalized Differential Evolution (GDE) algorithm for scheduling long-range activities on NASA's Deep Space Network. Initial results show that significant speedups can be achieved, but that performance does not necessarily improve as more cores are utilized. We describe our preliminary results and some initial suggestions from parallelizing the GDE algorithm. Directions for future work are outlined.

Introduction

In the context of scheduling NASA's network of deep space communications antennas, shared among dozens of users, the use of multi-objective optimization provides some unique capabilities. It enables the explicit representation of user and system objectives, and therefore supports their tradeoffs, which can be user-to-user, or can be among users and the network operations as a whole. As part of a JPL project to upgrade the Deep Space Network (DSN) scheduling applications and databases, we have been investigating multi-objective optimization including problem representation, algorithms, performance, and user interfaces. In this paper we describe the overall long-range scheduling problem, and then specifically some recent progress on a parallel multi-core evolutionary algorithm.

There are a variety of approaches to parallelizing search algorithms, and specifically evolutionary algorithms. In the latter, a population of candidates is evolved to approximately sample the Pareto frontier. We have been particularly interested in multi-core parallelism that would allow the use of readily available hardware in a higher performance mode. In the following we first describe the scheduling domain of interest – NASA's Deep Space Network (DSN) – and how it is planned and scheduled. We then describe the evolutionary algorithm we are using and how we have applied it to the DSN application domain, and our design of a parallel multi-core version based on the Java *ForkJoin* library classes. This is followed by a description of our experiments and our initial results, including performance improvements of nearly an order of magnitude speedup on commodity hardware. Finally we conclude with a brief description of plans for future work.

Long-Range Scheduling for NASA's DSN

NASA's Deep Space Network (DSN) (Imbriale 2003) is comprised of a set of large (34m and 70m diameter) antennas and the associated equipment required to communicate with spacecraft, from those in high Earth orbit to the most distant man-made objects. These antennas are situated at three Deep Space Communications Complexes (DSCC), spaced roughly equally in longitude, to provide round the clock coverage for missions anywhere in space. Although capabilities vary from one antenna to another, and one complex to another, overall the DSN provides a range of S, X, and Ka band up- and downlink services to both NASA and international partner missions. These services include support for spacecraft telemetry, command, and tracking, as well as radio science, radio astronomy and Very Long Baseline Interferometry (VLBI), radar, and calibration.

Currently the DSN supports 37 spacecraft or service users, counting all those with regular requirements for scheduled time on any antenna. The mission users span a wide range of distance and orbit type: high earth orbit, lunar orbit, solar orbit, probes at Mercury, Venus, Mars, and Sat-

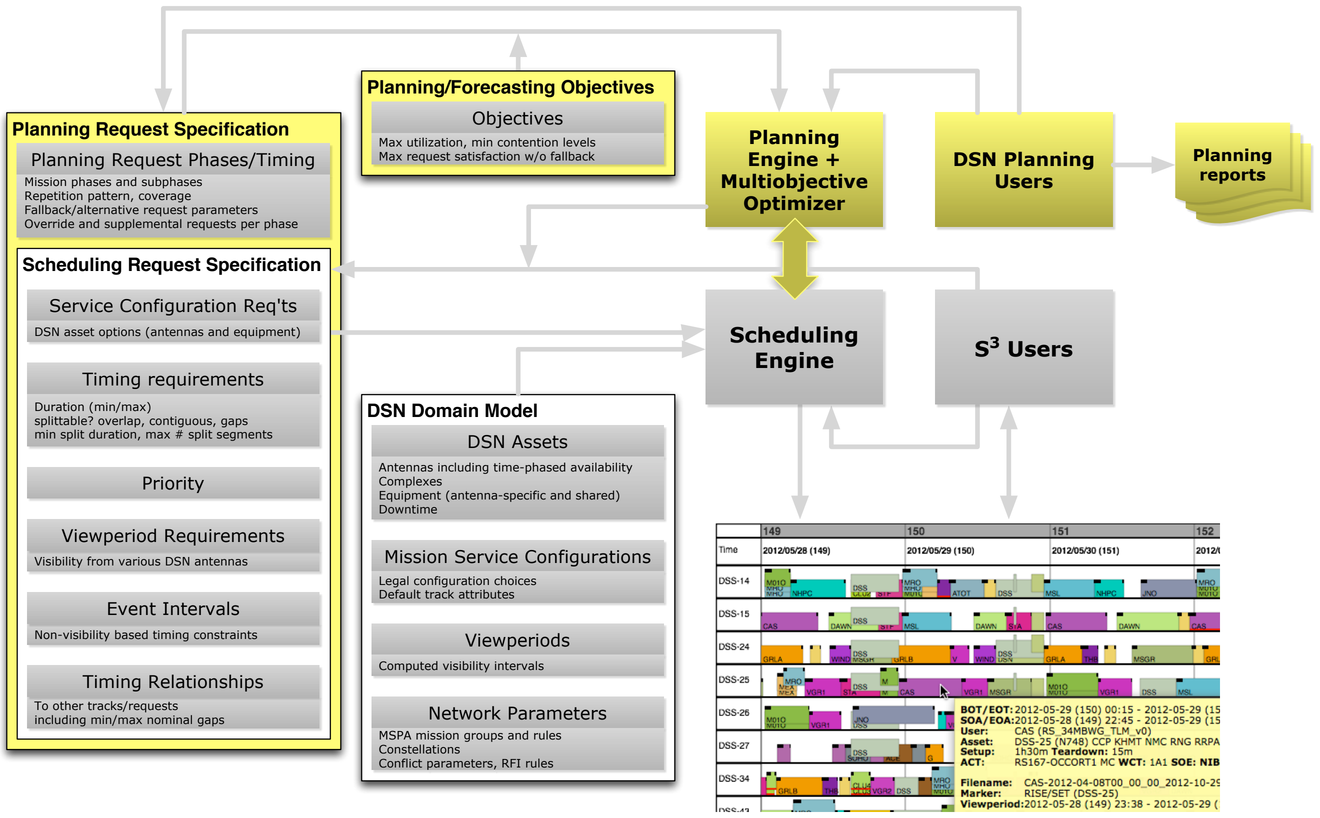
Planning Request Specification
Planning Request Phases/Timing
Mission phases and subphases
Repetition pattern, coverage
Fallback/alternative request parameters
Override and supplemental requests per phase
Scheduling Request Specification
Service Configuration Req'ts
DSN asset options (antennas and equipment)
Timing requirements
Duration (min/max)
splittable? overlap, contiguous, gaps
min split duration, max # split segments
Priority
Viewperiod Requirements
Visibility from various DSN antennas
Event Intervals
Non-visibility based timing constraints
Timing Relationships
To other tracks/requests
including min/max nominal gaps
Planning/Forecasting Objectives
Objectives
Max utilization, min contention levels
Max request satisfaction w/o fallback
DSN Domain Model
DSN Assets
Antennas including time-phased availability
Complexes
Equipment (antenna-specific and shared)
Downtime
Mission Service Configurations
Legal configuration choices
Default track attributes
Viewperiods
Computed visibility intervals
Network Parameters
MSPA mission groups and rules
Constellations
Conflict parameters, RFI rules
Planning Engine + Multiobjective Optimizer
DSN Planning Users
Planning reports
Scheduling Engine
S³ Users
Time
DSS-14
DSS-15
DSS-24
DSS-25
DSS-26
DSS-27
DSS-34
BOT/EOT: 2012-05-29 (150) 00:15 - 2012-05-29 (15
SOA/EOA: 2012-05-28 (149) 22:45 - 2012-05-29 (15
User: CAS (RS_34MBWG_TLM_v0)
Asset: DSS-25 (N748) CCP KHMT NMC RNG RRPA
Setup: 1h30m Teardown: 15m
ACT: RS167-OCCORT1 MC WCT: 1A1 SOE: NIB
Filename: CAS-2012-04-08T00_00_00_2012-10-29
Marker: RISE/SET (DSS-25)
Viewperiod: 2012-05-28 (149) 23:38 - 2012-05-29 (

Figure 1. Block diagram of LAPS showing commonality with the DSN mid-range scheduling software. New elements for LAPS include Planning Requests (left), an explicit representation of objectives, a new planning engine, and a user interface and reporting mechanism (top). The illustrated user interface (lower right) is from the S^3 mid-range scheduling system web application.

urn (and en route to Jupiter and Pluto/Charon), and the asteroids, out to the two Voyager spacecraft in interstellar space. Ground-based users conduct radio science and radio astronomy using the antennas, including coordinated programs with international partners. Other activities that must be scheduled include routine and special maintenance, calibration, engineering, and test activities. The collected set of DSN users imposes a very wide range of usage requirements on the network due to differing designs and operating modes. Some users require occasional contacts of only a few hours per week, but this ranges up to continuous coverage during certain mission phases, such as post-launch and during critical mission events. At the present time, a typical week includes about 500 scheduled activities on the antennas of the three DSN complexes.

Scheduling the DSN

The DSN scheduling process (Johnston et al. 2012) consists of three phases, which do not have sharply defined boundaries:

Long-Range Planning and Forecasting. Long-range planning is based on user-provided high-level requirements, spanning from the present day to the anticipated end of mission. Long-range planning has several major purposes:

- studies and analyses: periods of particular interest or concern are examined to determine where there is likely contention among missions, for example around launches or critical mission events (maneuvers, planetary orbit insertion or landings), or when construction of a new DSN antenna is under investigation
- downtime analysis: identifying periods of time when necessary antenna or other maintenance can be scheduled, attempting to minimize the impact on missions
- future mission analysis: in proposal phase, missions can request analysis of their proposed DSN coverage as part of assessing and costing proposals for new missions

The time range for long-range planning is generally six months or more into the future, frequently as much as years.

Mid-Range Scheduling. The mid-range scheduling phase is when detailed user requirements are specified, integrated, negotiated, and all tracking activities finalized in the schedule. Starting at roughly 4-5 months before execution, users specify their detailed scheduling requirements on a rolling weekly basis. These requirements include:

- detailed tracking time and services required
- constraining time intervals and relationships
- visibility constraints
- flexibilities

Further discussion of the nature of these requirements and flexibilities is provided elsewhere (Johnston et al. 2012). The mid-range scheduling process includes an automated phase where conflicts are detected and minimized by the DSN Scheduling Engine, after which users collaboratively negotiate any remaining conflicts using the S^3 web application.

Near Real-time Scheduling. The (near) real-time phase of DSN scheduling starts roughly eight weeks from execution and includes the period through execution of all the scheduled activities. Late changes may occur for various reasons, and these can impact the mid-range phase as well.

Loading Analysis and Planning Software (LAPS)

The DSN has undertaken an overall unification and simplification of the scheduling software systems (Johnston et al. 2012) of which the first increment has been operational since June 2011. This is called the Service Scheduling Software (SSS, or S^3) and has initially been applied only to the mid-range process. The long-range component is in development now, designated the Loading Analysis and Planning Software, or LAPS.

LAPS builds on the mid-range scheduling component that is already deployed operationally. However, it must additionally deal with some other factors, such as:

- numerous and sometimes intrinsic sources of uncertainty, including:
  - unpredictable spacecraft locations for some missions and trajectory types, leading to uncertainties in visibility times from the different DSN antennas
  - unknown science targets beyond some time horizon in the future
  - uncertainties in the mission set, due to funding changes, launch date changes, or mission extensions
- optimization criteria and scenarios that differ from mid-range, where the main objectives are to minimize conflicts in the schedule and violations of user requirements; for long-range a variety of other objectives may come into play, including:
  - identifying best times to schedule extended downtime for preventive maintenance, minimizing the impact on active missions
  - identifying best times to schedule special flexible but resource intensive operations, such as reference frame calibration activities
  - maximizing the satisfaction of requirements where, due to contention, not all requirements can be satisfied across the entire DSN user base

In addition, long range planning needs to provide information to mission planners about where contention with critical events may occur, so that this can be taken into account as early as possible in each mission's planning process. In many cases this needs to be provided during the mission proposal phase when, for both feasibility and costing, it is necessary to map out DSN allocation needs to some preliminary level of accuracy.

Finally, long-range planning needs to support specification of a more abstract type of requirement with less detail than would be acceptable in mid-range. This serves two purposes: it represents at a coarse level some of the uncertainty in requirements, and it makes it easier to specify "what if" alternative scenarios.

Figure 1. shows an overall block diagram of the LAPS software, emphasizing commonality with the mid-range system. One of the major new components is the planning engine and optimizer, the subject of further discussion below.

Multi-Objective Scheduling with an Evolutionary Algorithm

In the DSN scheduling problem there are numerous objectives from the missions and other users, as well as from overall system operations. Conventional approaches to schedule optimization would combine these into a single objective value to optimize. However, there is a significant drawback to this approach, in that it pre-specifies the tradeoffs among the different objectives. That is, for any given value of the combined single objective, the results of optimization will not distinguish between cases where any one mission's objectives are satisfied anywhere from fully to not at all. From the perspective of a mission or user, such an optimization process is at least cause for concern.

An alternative approach is to use multi-objective optimization techniques, which retain the information in separate objectives until the end of the optimization process: see e.g. (Johnston 2006; Johnston 2008; Johnston & Giuliano 2011a; Giuliano & Johnston 2008) and references therein. When there remains contention to be resolved, a multi-objective approach will provide explicit information about the tradeoffs involved. Such visibility is important to users who may be required to compromise on the achievement of their objectives.

Among techniques developed to solve multi-objective optimization problems, evolutionary algorithms (Deb

2001) have become popular for a variety of reasons, including their capability to deal with objectives that are not mathematically well behaved (e.g. discontinuous, non-differentiable). A solution is called *Pareto-optimal* when no improvement can be made to one objective which does not make worse at least one other objective. Because evolutionary methods maintain a *population* of solutions, they can trace out the *Pareto frontier*, an approximation to the entire set of Pareto-optimal solutions. Evolutionary algorithms also lend themselves to parallelization, which can be an advantage for large problems. Two important performance characteristics of a multi-objective evolutionary algorithm are *convergence* to the Pareto frontier, and *diversity* so as to maximally sample the frontier.

The multi-objective optimization problem we consider is that of minimizing a set of M objectives subject to K constraints:

$$\text{minimize: } \{f_i(\vec{x})\},\ i = 1 \ldots M$$
$$\text{subject to: } (g_j(\vec{x}))^T \le 0,\ j = 1 \ldots K$$

Here $\vec{x}$ represents a vector in decision space of dimension D. Where necessary below, we refer to the i^{th} member of the population at generation g as $\vec{x}_{i,g}$.

An evolutionary algorithm formulation for multi-objective optimization consists of a population of N solution candidates. With each step (generation) of the algorithm, we evolve the population according to method-specific rules for crossover and mutation. Crossover combines elements of parent solution candidates to generate a new offspring candidate for evaluation. Mutation introduces randomized variation in the offspring.

Depending on the procedure, following the crossover and mutation steps, the size of the population may have increased. If so, it is reduced back to N through a selection procedure. The overall process repeats, with a cutoff generally determined by a specified maximum number of generations G^{max}.

Generalized Differential Evolution (GDE3)

GDE3 (Kukkonen & Lampinen 2005) is an algorithm that uses Differential Evolution (DE) (Storn & Price 1997) as the basis for crossover and mutation operations, and non-dominated sorting and crowding distance in a manner similar to NSGA II (Deb et al. 2002). DE is an evolutionary algorithm for optimization, initially developed in a single objective context. DE is defined on real-valued decision spaces. It generates offspring through the following procedure:

1. For each parent $\vec{x}_i$, select three distinct population members $\vec{x}_{r_1}, \vec{x}_{r_2}, \vec{x}_{r_3}$, all different and different from parent
2. Calculate a trial vector $\vec{y}_i$ as:
   $$\vec{y}_i = \vec{x}_{r_1} + F \cdot (\vec{x}_{r_2} - \vec{x}_{r_3})$$
   where F is a scaling factor
3. Modify the trial vector by binary crossover with parent with probability CR
4. Calculate the objective and constraint values for the trial vector

The result is compared with the parent as follows:

- in the case of *infeasible* vectors, the trial vector is selected if it weakly dominates the parent vector in constraint violation space, otherwise the parent vector is selected
- in the case of *feasible* and *infeasible* vectors, the feasible vector is selected
- if *both* vectors are feasible, then the trial is selected if it weakly dominates the parent in objective space; if the parent dominates the trial, then the parent is selected, and if neither dominate, then both are selected

The selected vectors may constitute a set of size $>N$, in which case the population size is reduced by non-dominated sorting, using crowding distance as a tie-breaker in order to bias selection towards better coverage of the Pareto frontier (Deb et al. 2002).

GDE3 is appealing for several reasons: it provides a natural treatment of the K constraints, while reducing to standard DE when the number of objectives M=1. The treatment of constraints makes it straightforward to change constraints into objectives when investigating overconstrained problems. GDE3 performs very well in initial comparisons with other algorithms, and does not introduce any additional control parameters beyond F and CR from the original formulation of DE.

DSN Long-Range Plan Optimization

We have adapted the MUSE implementation of GDE3 (Johnston & Giuliano 2009; Johnston & Giuliano 2011b) as the multi-objective algorithm for the LAPS long-range planning engine. In our current experimental version, decision variables represent the relative priority of each DSN user, binned into an adjustable interval duration. For the experiments report here, we used one-week intervals. As an additional decision variable we allow requirements to be tagged as *nominal*, *reduced*, or *minimal*, so that the loading analysis can automatically consider tradeoffs with reduced requirement scenarios. In our current experiments, however, all requirements were left at nominal.

For planning objectives we are currently using a very simple set of two minimization objectives:

- O_1: unscheduled requirement time, i.e. total time specified as required by users, but not able to fit into the plan
- O_2: total track duration scheduled on all antennas

Objective O_2 may seem unusual, but for a multi-objective formulation it is very informative: for a given level of unscheduled requirement time O_1, a better schedule is one with more open antenna time, which would allow for additional new requirements to be met.

Our test problem includes requirements for a 16 week time span in 2012, totaling just over 3,000 requirements requesting 962 days of time on the 13 large (34m and 70m) DSN antennas. Typical scheduling runs place over 4,500 activities on the schedule during these 16 weeks. There is less than 10% oversubscription, but there is no feasible schedule with all requirements met.

Parallelizing the GDE3 Algorithm

The GDE3 algorithm includes a loop that generates *N* new trial population members as described above; each new member requires the evaluation of the objective function values, which is the most time-consuming part of the algorithm. However, because each such evaluation is independent of the others, it is possible to parallelize this generate/evaluate portion of the algorithm. Each one of the *N* trials can be evaluated independently of the others, provided no dependencies are introduced by the evaluation mechanism. Once all *N* have been evaluated and selected (or not), the next step is to sort and reduce the population back to size *N* for the next generation.

An algorithm with this structure lends itself well to decomposition into multiple independent computational tasks. Such tasks can be managed by execution frameworks that have been developed for use in many languages and systems. Since LAPS is implemented in Java, we have investigated Java capabilities that support parallelism. The Java language itself includes thread primitives that could be used to implement parallel executing tasks suitable for this problem. However, the latest release of the Java language, Java 7, includes a *ForkJoin* framework explicitly aimed at facilitating multi-core implementation of algorithms in Java. We have evaluated this new framework and found that it provides a straightforward mechanism for parallelizing the GDE3 algorithm, as follows:

- (Fork) For each generation, create *N* Java *Callable* tasks that implement steps 1-4 above, including time-consuming the objective calculation
- (Join) When all *N* tasks have completed, perform the population reduction as needed, then prepare for the next generation

The *ForkJoin* implementation in Java defaults to creating the maximum number of parallel activities supported by the hardware platform. However, it provides parameterization so that the user can run with a specified degree of parallelism. When set to a value of 1, the framework runs essentially serially. We have used this parameterized capability to evaluate the increasing benefit of more cores, and to see where diminishing returns sets in.

Results

We have conducted a series of experiments on three different multi-core computers described in Table 1. Two of these are end-user systems such as may be found in a typical high-end office environment. The Linux server is a more expensive ($25K) rack-mounted server such as might be found in a typical data center.

System	Description	Processor	RAM	cores
A	Laptop – MacBook Pro (2012 retina display)	2.7 GHz Core i7	16 GB	8
B	Desktop – Mac Pro (2011)	2x 2.93 GHz Xeon X5670	64 GB	24
C	Linux server Sunfire x4450 (2009)	4x 2.66 GHz Xeon X7460	128 GB	24

Table 1. The experimental system configurations.

For an initial set of experiments, we modified the serial GDE3 algorithm to make use of the Java 7 *ForkJoin* framework and compared run times when *ForkJoin* was constrained to use just a single core. We found the results to be virtually identical, and so as a basis of comparison we use the "1-core" results to measure speedup.

The effect of adding additional cores for system A (laptop) is shown in Figure 2A. For these runs, the maximum heap size was set at 12GB. Going from one to two cores led to a speedup factor of 1.8, but the improvement leveled out quickly and the maximum speedup (with 8 cores) was a factor of 3.0. In contrast, the desktop system (Figure 2B), with a total of 24 cores, continued to show noticeable improvement until about half of these were utilized: the overall speedup factor topped out at about 7.3.

This uniform improvement with increasing number of cores on systems (A) and (B) did *not* hold on the third system (C) however. Figure 2C shows the results of executing the test on the Linux server system (C). This server has somewhat older and slower processors and so took longer per generation than the other two systems by about a factor of 2 to 3. Increasing the number of cores to 12 showed a similar proportional improvement to system (B), about 7.3x speedup, but adding more cores initially made little difference, then performance *worsened* dramatically: using 24 cores was about the same as 2! This effect, while unexpected, has been reported as a consequence of memory bandwidth limitations (Singer 2009), which is consistent with it appearing on the oldest of our test systems.

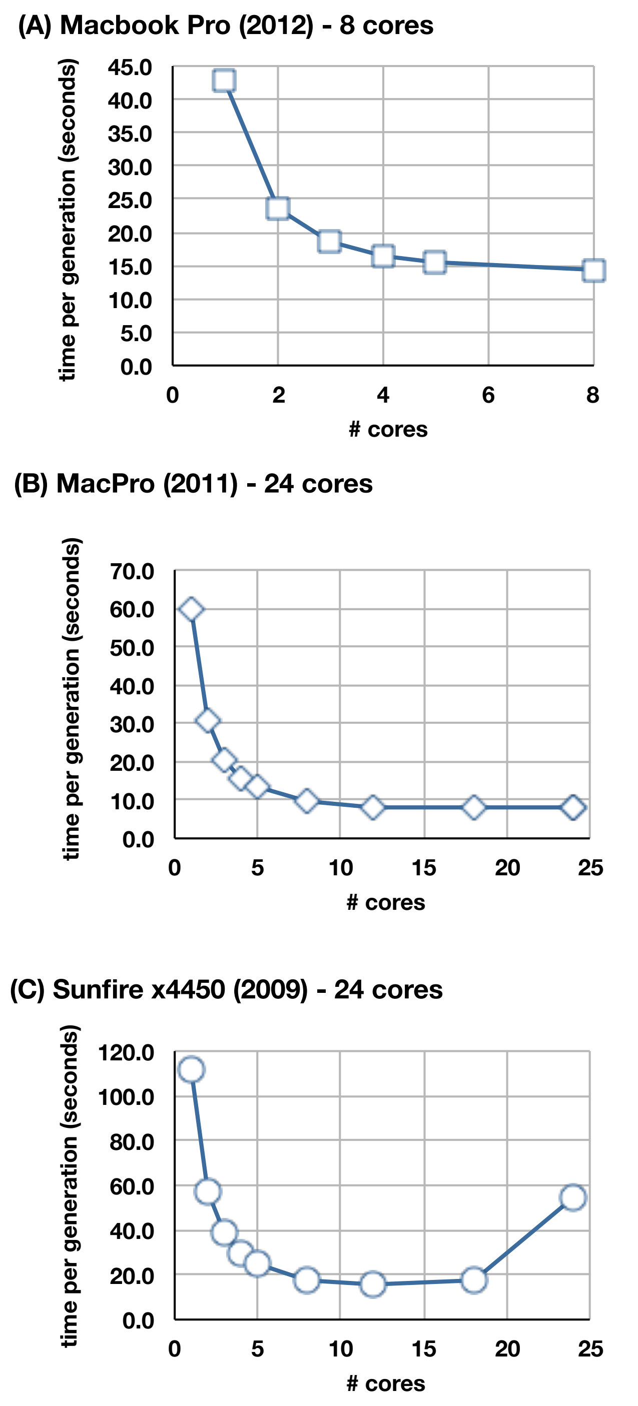

Figure 2: (A) Time per generation speedup for the laptop system vs. number of cores. The maximum speedup is about a factor of 3; (B) same for the desktop system vs. number of cores. The maximum speedup is about 7.3x but levels out at about 12 cores; (C) same for the Linux server system vs. number of cores. The maximum speedup is also about 7.3x for 12 cores, but worsens as more cores are used to the point that 24 cores are not much better than 2.

We conducted some investigation of the effect of maximum heap size on our results. First, it is necessary to point out that for this particular problem, the objective calculation requires the generation of a large number of transient Java objects that need to be reclaimed from the heap when no longer needed. Java has very efficient garbage collection mechanisms, but it does introduce some coupling among the multiple independent tasks that they are sharing the Java heap. For the laptop system A, we found that running with a 4GB heap size (vs 12GB) caused a slowdown of about 25% in runtime for the most parallel runs, but a negligible slowdown for small numbers of cores. This is to be expected as the large number of parallel tasks tends to consume a correspondingly large amount of heap space. On the desktop system, we found that running with a very large heap (48GB) led to only a 13% improvement over runs with a 12GB heap, the same as the maximum on the laptop system A. Clearly it is important to allocate enough heap for the multiple parallel tasks to run with minimal contention, but providing much more heap does not make a significant difference.

To help those interested in applying this *ForkJoin* mechanism to their own algorithms, we offer the following suggestions:

- design the objective function evaluation with minimal writing to shared data structures, and be sure that any such access is synchronized
- allocate only a single *ForkJoin* instance (which creates the worker threads as a relatively expensive operation) and make multiple calls to invoke the parallel tasks
- ensure sufficient heap size to allow the desired number of parallel tasks to coexist without being close to the maximum heap – this will minimize slowdowns due to garbage collection
- assess performance as more cores are added to ensure that memory bus contention does not negate the gains arising from increased parallelism

Relationship to Previous Work

Previous work related to ours falls into several categories. First, there has been a good deal of general investigation of parallel evolutionary algorithms of all kinds. The ability to decompose computations on different population members has been explored in a wide range of parallel algorithms, for example (Alba & Tomassini 2002; Luque et al. 2005; Nedjah et al. 2006). In addition, there has been specific work on parallelizing the single objective form of differential evolution (DE), such as described in (Tasoulis et al. 2004) and (Storn 2008). Finally, some researchers have addressed single objective multi-core parallel DE with algorithm variants, on test problem sets (Tagawa & Ishimizu 2010) and as applied to medical image registration (Cao et al. 2009). The parallelization of single objective DE and multi-objective GDE3 follows very similar lines, but we are not aware of any published results for parallelized multi-objective DE.

Conclusions

In this paper we have described the application of an evolutionary multi-objective algorithm to the DSN long-range scheduling problem, and of our initial experience in parallelizing this algorithm to take advantage of standard commercial multi-core hardware. We have measured speedups between 3x and 7x on laptop and desktop systems, respectively, which represent a major performance boost. However, we also encountered a system where increasing parallelism did not continue to improve performance.

The next stage in our investigation of these algorithms is to explore how solution quality varies with parameter (population size, number of generations, etc.), and our parallel multi-core implementation will enable a much more efficient investigation.

Additionally, we are investigating:

- the use of local search techniques to further improve generated schedules
- how alternative decision variable representations affect solution quality
- user presentation techniques to enable visibility into tradeoffs among objectives

Acknowledgments

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Alba, E. & Tomassini, M., 2002. Parallelism and evolutionary algorithms. *IEEE transactions on evolutionary computation*, 6(5), pp.443–462.

Cao, G., Luo, L. & Rong, C., 2009. Multicore-based parallelized differential evolution for medical image registration. *MIPPR 2009: Medical Imaging*, 7497, p.91.

Deb, K., 2001. *Multi-Objective Optimization Using Evolutionary Algorithms*, New York: John Wiley & Sons.

Deb, K. et al., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2), pp.182–197.

Giuliano, M. & Johnston, M.D., 2008. Multi-Objective Evolutionary Algorithms for Scheduling the James Webb Space Telescope. In International Conference on Automated Planning and Scheduling (ICAPS). Sydney, Australia.

Imbriale, W.A., 2003. *Large Antennas of the Deep Space Network*, Wiley.

Johnston, M.D., 2008. Deep Space Network Scheduling Using Multi-Objective Optimization With Uncertainty. In SpaceOps. Heidelberg, Germany.

Johnston, M.D., 2006. Multi-Objective Scheduling for NASA's Deep Space Network Array. In International Workshop on Planning and Scheduling for Space (IWPSS-06). Baltimore, MD: Space Telescope Science Institute.

Johnston, M.D. & Giuliano, M., 2011a. Multi-Objective Scheduling for Space Science Missions. *Journal of Advanced Computational Intelligence and Intelligent Informatics (JACII)*, 15(8), pp.1140–1148.

Johnston, M.D. & Giuliano, M., 2011b. Multi-Objective Scheduling for the Cluster II Constellation. In 6th International Workshop on Planning and Scheduling in Space (IWPSS). Darmstadt, Germany.

Johnston, M.D. & Giuliano, M., 2009. MUSE: The Multi-User Scheduling Environment for Multi-Objective Scheduling of Space Science Missions. In IJCAI Workshop on Space Applications of AI. Pasadena, CA.

Johnston, M.D. et al., 2012. Automating Mid- and Long-Range Scheduling for NASA's Deep Space Network. In SpaceOps 2012. Stockholm, Sweden.

Kukkonen, S. & Lampinen, J., 2005. GDE3: The Third Evolution Step of Generalized Differential Evolution. In The 2005 Congress on Evolutionary Computation. p. 443.

Luque, G., Alba, E. & Dorronsoro, B., 2005. Parallel Genetic Algorithms. In *Parallel metaheuristics: A new …*. A New Class of Algorithms. Hoboken, NJ, USA: John Wiley & Sons, Inc., pp. 105–125.

Nedjah, N., Alba, E. & de Macedo Mourelle, L., 2006. *Parallel Evolutionary Computations* Studies in Computational Intelligence, Springer.

Singer, N., 2009. More chip cores can mean slower supercomputing, Sandia simulation shows. Available at: https://share.sandia.gov/news/resources/news_releases/more-chip-cores-can-mean-slower-supercomputing-sandia-simulation-shows/.

Storn, R., 2008. Differential Evolution Research – Trends and Open Questions. In *Advances in differential evolution*. Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–31.

Storn, R. & Price, K., 1997. Differential Evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11, pp.341–350.

Tagawa, K. & Ishimizu, T., 2010. Concurrent implementation of differential evolution. In 10th WSEAS International Conference on Systems Theory and Scientific Computation (ISTASC '10). Tapei, Taiwan: World Scientific and Engineering Academy and Society (WSEAS), pp. 65–70.

Tasoulis, D.K. et al., 2004. Parallel differential evolution. *Congress on Evolutionary Computation (CEC 2004)*, 2.