Ranger Optimization

Drew Dolgert

May 19, 2010

Contents

1 Introduction **1**
1.1 The Ask . . . 2
1.2 Exercise: Allocation MadLib . . . 4

2 Numerical Libraries on Ranger **4**
2.1 Discussion: Find a Relevant Math Library . . . 4
2.2 Exercise: Compare Libraries and Hand-written Code . . . 5

3 Compiler Optimization **6**
3.1 Common Compiler Options . . . 6
3.2 Exercise: Test Numerically-intensive Code . . . 7

4 Scalability **7**
4.1 Exercise: Excel Demo of Fluent Model . . . 7
4.2 Exercise: Measure Strong Scaling of Fourier Transform . . . 8
4.3 Exercise: Measure Memory Usage of Fourier Transform . . . 8
4.4 What Machines Are Appropriate for Your Work? . . . 8

Contents:

1 Introduction

This document accompanies a talk on optimization and scalability on the Ranger cluster at the Texas Advanced Computing Center.

The goal of the talk is to help researchers use the Ranger system efficiently. Ranger is a large, distributed system made to run parallel, distributed code. We usually address how to accelerate applications in two steps, making code faster on one node, and making code faster in parallel across several nodes. The former is optimization, the latter scalability.

You should have the following skills by the end:

- Know how to write an allocation request.
- Know when to focus your efforts on writing code, compiling it, choosing a new algorithm, or finding another computer.
- Predict a parallel algorithm's effect on your program's speed from its complexity.

Computational simulations are built on a series of choices.

1. What *model* would inform my research?
2. Which set of *algorithms* could express this model on a computer?
3. How should I *implement* those algorithms in code?
4. What *compiler* will best interpret that code, and with what hints?
5. On what *runtime environment* will this executable run?

These questions don't happen in order, though. You already know what model you want to compute and have probably chosen what algorithms, even what applications, you want to use to do that computation. On the other hand, you have a certain set of computers or clusters available, from Ranger to your laptop. It's the in-between steps, of compilation, implementation, and algorithm, where you get to make the most choices.

We cannot cover, in this talk, how to write good code. If writing code is crucial to your work, you can likely already construct the computational equivalent of a NASCAR engine. What we will do is remind you Formula 1 engines exist and take you on a tour equivalent libraries and applications on Ranger that might help you avoid doing your own coding.

Modern compilers can now outmatch all but the finest of hand-tuned efforts at constructing machine code. The key to using a compiler is to help it understand your code and inform it of all the capabilities of the architecture on which you will run. Once you know the options, finding the best compiler flags will still be an experimental process.

Scalability is whether an algorithm can remain efficient when run on many nodes. While we won't cover algorithm choice, we will learn how to measure scalability and how to use it in an argument about whether a particular algorithm is a good choice to run on Ranger.

1.1 The Ask

You can't get a research allocation on Ranger unless you can explain the scalability of your program, so how do you do that?

One of the Cornell groups working on Ranger was kind enough to let use use part of a draft of their winning application for a research allocation on Ranger.

The first section of the application is about the science, the second an explanation of what computations they do. Let's take a look at the third section, Preliminary Results. They offer two figures, here labeled Fig. 1 and Fig. 2. Their text follows.

> This section presents preliminary results obtained with stand-alone usage of all three components of our LES/FDF/ISAT methodology along with some information on their scalability. The scalability studies have been performed on TACC Ranger and on our CATS cluster.
>
> The parallel performance of the LES code in a variable-density flow has been studied for the case of a simple, non-premixed jet flame at Re=22,400 on a cylindrical grid with resolution 128 x 96 x 64. A flamelet table has been employed to provide the chemical parameterization of the filtered density as a function of mixture fraction. Preliminary scalability results are shown in Figs. 1 (a,b). It is seen that for the chosen grid resolution the LES code exhibits linear scalability up to 128 processors and reasonable scalability up to 256 processors. We expect improvements in scalability with increasing problem size.

This application is explicit about time from a previous application having been used to do scaling studies. The studies shown are for *strong scaling*, which means they examine how the solution time varies with the core count for fixed problem size. The last sentence of the application points

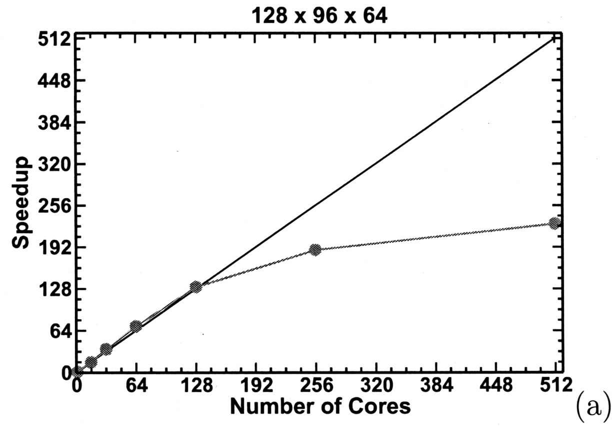

Figure 1: Speedup graph for example code.

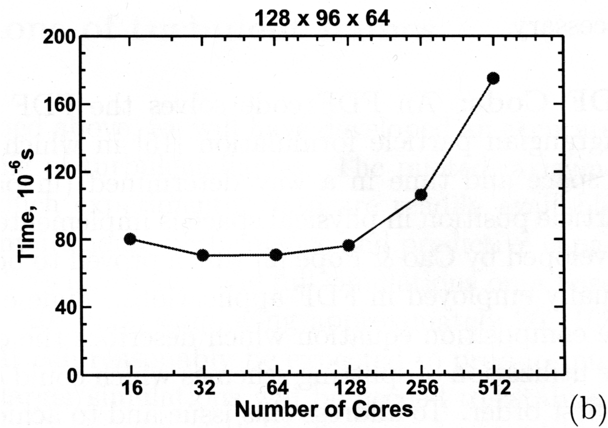

Figure 2: How wall time scales with number of cores, but the wall time is multiplied by the number of cores.

out that *weak scaling*, meaning the change in wall time for increasing problem size, should remain efficient to much larger core counts.

1.2 Exercise: Allocation MadLib

Goal: Know how to write this section of an allocation proposal.

Demonstration of the current scaling capabilities of your code can be fairly straightforward. With the data given for a sample FFTW2 code, which solves a three-dimensional real transform, try filling in the MadLib below.

> The main algorithm of our application is a spectral code which relies on fast fourier transforms. We use the FFTW2 library because it runs over MPI and is optimized for our target platform.

problem size	wayness	cores	time
1024	16	16	28.17
1024	16	16	28.47
1200	16	16	50.01
1200	16	32	30.49
1200	16	64	19.18
1200	16	128	13.38
1200	16	256	12.79
1200	16	256	13.12
1200	16	256	16.97
1200	16	512	24.16
1200	16	1024	44.74
2048	4	256	73.38

> We tested the application on the Ranger system with 16 cores per node and Infiniband among nodes. For our smallest runs, with ___ cores, we see a wall time of ___. For larger runs, maintaining a problem size of 1024 cubed, the efficiency drops below 60% between ___ and ___ cores.
>
> Given these results, we feel we can compute our system for one microsecond of simulation time using 10000 iterations at ___ cores, within __ SUs.

Why do you think there are no 1way jobs for a 1200 cubed problem size?

2 Numerical Libraries on Ranger

2.1 Discussion: Find a Relevant Math Library

Pick from the list above a library that you don't already know. See if you can find something interesting about it from its documentation. Compare with your neighbor, and we'll share that with the group. Typical questions are

- Is it multi-threaded?
- Can it run distributed (which means across nodes, which means with MPI)?
- What languages can use it?

There are a lot of domain-specific applications on Ranger. Let's just look at some of the math libraries.

- acml - AMD Core Math Library, lapack.

- arpack - Fortran subroutines to solve large scale eigenvalue problems.
- fftw2 - A fast, free C FFT library; includes real-complex, multidimensional, and parallel transforms. Version 2 can use MPI.
- fftw3 - A fast, free C FFT library; includes real-complex, multidimensional, and parallel transforms.
- glpk - GNU linear programming kit.
- gotoblas - Hand-tuned Basic linear algebra subroutines.
- hypre - library for solving large, sparse linear systems of equations
- mkl - Intel Math Kernel Library.
- mpfr - C library for multiple-precision floating point.
- numpy - Tools for numerical computing on Python.
- petsc - data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations.
- plapack - Parallel linear algebra matrix manipulations.
- pmetis - parallel graph partitioning.
- scalapack - linear algebra routines using MPI.
- scalasca - open-source analysis of parallel applications.
- slepc - Scalable library for eigenvalue problems.
- sprng - scalable parallel psuedo random number generation.
- trilinos - algorithms for large-scale, complex multi-physics engineering and scientific problems.

2.2 Exercise: Compare Libraries and Hand-written Code

Goal: Understand when to use numerical libraries in your code.

Unpack it with:

```
$ cd
$ tar zxf ~train400/opti.tgz
```

You will find an `opti` directory containing two sets of code in subdirectories. You can compile both by typing `make` in the `opti` directory. The executables will sit in `opti/bin`, along with a batch script that runs those executables.

- The `nr` version uses Numerical Recipes.
- The `gsl` version uses Gnu Scientific Libraries.

1. Add `gsl` to the currently-loaded modules.
2. Ensure `pgi` is the current compiler, again using the module command.
3. Type `make` in the ~/ludecomp directory.
4. Submit the job.sge script with your account in it. One way to do this is `qsub -A 20100519HPC job.sge`. The results of your runs will be in results.txt.

There are two versions of the same code, all reasonable. They are in nr.c, gsl.c. A third version, using lapack.c, isn't finished, but you can take a look. Evaluate each one on these criteria:

1. How many lines of code did it take to implement the solution?
2. How fast does it run? Look at results.txt when it finishes.
3. For each version, how hard would it be to use an alternative algorithm, for instance, to use an iterative solver or to solve a sparse matrix?
4. What are the capabilities of each code for running in parallel, meaning are they multi-threaded? Can they run distributed?

3 Compiler Optimization

3.1 Common Compiler Options

Standard Choices are:

- PGI: `-O3 -fast tp barcelona-64 Mipa=fast`
- Intel: `-O3 -xW -ipo`

Don't exceed -O2 without checking whether your output is correct.

PGI

- -O3 - Performs some compile time and memory intensive optimizations in addition to those executed with -O2, but may not improve performance for all programs.
- -Mipa=fast,inline - Creates inter-procedural optimizations. There is a loader problem with this option.
- -tp barcelona-64 - Includes specialized code for the barcelona chip.
- -fast - Includes: -O2 -Munroll=c:1 -Mnoframe -Mlre -Mautoinline -Mvect=sse -Mscalarsse -Mcache_align Mflushz
- -g, -gopt - Produces debugging information.
- -mp - Enables the parallelizer to generate multi-threaded code based on the OpenMP directives.
- -Minfo=mpi,ipa - Provides information about OpenMP, and inter-procedural optimization.

Intel

- -O3 - More than O2, but maybe not faster
- -ipo - Creates inter-procedural optimizations.
- -vec_report[0|..|5] - Controls the amount of vectorizer diagnostic information.
- -xW - Includes specialized code for SSE and SSE2 instructions (recommended).
- -xO - Enables the parallelizer to generate multi-threaded code based on the OpenMP directives.
- -fast - Includes: -ipo, -O2, -static DO NOT USE – static load not allowed because only dynamic loading is allowed.
- -g -fp - debugging information produced.
- -openmp - Enable OpenMP directives

- -openmp_report[0|1|2] - OpenMP parallelizer diagnostic level.

3.2 Exercise: Test Numerically-intensive Code

Goal: Understand the process of optimizing compilation.

1. Return to the `~/ludecomp` directory.
2. Choose some compiler options from the list above. Set them in the `FFLAGS` variable in `makefile`, then type `make` and `qsub -A 20100519HPC job.sge`.
3. Iterate through a few choices of compiler and optimizations to see what is fastest, as shown in results.txt.
4. If you are a competitive sort, take a look at The Eric Chen Scoreboard.
5. What columns would you need in a chart to show your choices, and how do you choose what represents the results? Our main result is speed. What other choices could matter, depending on your problem?

4 Scalability

4.1 Exercise: Excel Demo of Fluent Model

Goal: Understand the different regimes of scaling and their dependence on latency and bandwidth.

For this exercise, we will look at an Excel spreadsheet that has analysis similar to what you would get from mpiP. We will use it to understand how Fluent might behave on different kinds of computers.

The Fluent program calculates fluid dynamics. It is a spectral code, like fourier transforms. The data, on the second page of the spreadsheet, show that, as you increase core count, Fluent sends disproportionately more messages and longer messages.

Given this, we assume:

- The main work speeds up according to the number of cores.
- Every message sent incurs a time penalty for latency.
- The time to send each message depends on its size, according to network bandwidth.

Download FluentEstimate.zip to the desktop. Double-click it and open the Excel 2007 file inside.

There are two graphs, one showing the speed on the right, and one showing the relative contribution of work, latency, and bandwidth on the left. The two sliders change latency and bandwidth.

1. Is our simplistic model enough to explain rougly at what core count the code becomes less efficient?
2. Infiniband and Gigabit ethernet have similar bandwidths, but Infiniband has much lower latency (under 2.3 microseconds on Ranger). Would using a cluster with this feature help run Fluent at higher core counts?
3. At what core count does the time to send messages become commensurate with the time to do the main work? How does this change as you lower latency? As you increase bandwidth?

4.2 Exercise: Measure Strong Scaling of Fourier Transform

Goal: Know how to estimate your code's scalability.

You will find code in `~/fftwmpi` which contains a parallel MPI FFTW2 code, the same one we used for the chart earlier. You have to add the modules for `pgi` and `fftw2` in order to compile it, which is done by typing `make`.

We ask, in this exercise, how much faster a parallel program can complete the same amount of work given an increasing number of processors. We need to choose an appropriate amount of work. That might normally be determined by the research, but here, we are limited by the amount of memory required to store an N x N x N array. A side length of 1200 seems to work well.

1. Run it on 1, 4, 16, 64, 256 processors by changing the makefile.
2. Make a chart on your local computer using a spreadsheet. It should be a log-log chart. Is the chart smooth?
3. You could fit the chart in order to estimate it.
4. What is the efficiency of running 16-way parallel on one node?
5. What is the efficiency of runnning 256-way parallel?

4.3 Exercise: Measure Memory Usage of Fourier Transform

Goal: See how memory per task changes with problem size and core count.

We ask how efficiently a parallel program can complete an *increasing* amount of work given an *increasing* number of processors. Most of the challenge here is finding what problem sizes fit into available memory. You will have to decrease wayness in order to get problems to fit in smaller core counts.

1. What is the largest size problem you can run 16way with 16 cores? What if you keep it 16way but increase core count? Note that the program prints its memory usage for the main work arrays.
2. Reduce wayness to 1way or 4way. How does it behave now at different core counts?
3. First make a standard log-log chart of the wall time at different core counts.
4. Look at memory usage per core for different core counts and problem sizes. This might be a more important metric for running this kind of code.

4.4 What Machines Are Appropriate for Your Work?

Look at the TeraGrid Portal Resource List. If you click on a name on the left, you will see a description of each machine. What are you looking for in those descriptions? It depends on what you need for your program, but the capabilities of those computers might also dictate what you imagine for your research.

- How much memory per core is on the machine? How much memory can a single process access?
- How many cores can you use for a multi-threaded portion of the code?
- Is the interconnect slow, like Gigabit Ethernet, or fast, like Infiniband?

- Does the interconnect have a special topology to support faster collective communications for large core counts?
- Are the filesystems parallel? What are the read and write rates?
- Will the chipset run regular Linux applications or is it special purpose, such as the Blue-Gene?
- Is the queue full for days?