STAR-P: High Productivity Parallel Computing

Ron Choy[*] Alan Edelman[*] John R. Gilbert[†] Viral Shah[†] David Cheng[*]

June 9, 2004

1 STAR-P

STAR-P [‡] is an interactive parallel scientific computing environment. It aims to make parallel programming more accessible. STAR-P borrows ideas from MATLAB*P [3], but is a new development. Currently only a MATLAB interface for STAR-P is available, but it is not limited to being a parallel MATLAB. It combines all four parallel MATLAB approaches in one environment, as described in the parallel MATLAB survey [2]: embarrassingly parallel, message passing, backend support and compilation. It also integrates several parallel numerical libraries into one single easy-to-use piece of software.

The focus of STAR-P is to improve user productivity in parallel programming. We believe that STAR-P can dramatically reduce the difficulty of programming parallel computers by reducing the time needed for development and debugging.

To achieve productivity, it is important that the user interface is intuitive to the user. For example, a large class of scientific users are already familiar with the MATLAB language. So it is beneficial to use MATLAB as a parallel programming language. Additions to the language are minimal in keeping with the philosophy to avoid re-learning. Also, as a design goal, our system does not distinguish between *serial* data and *parallel* data.

```
C = op(A,B)
print(C)
```

C will be the same whether A and B are distributed or not. This will allow the same piece of code to run sequentially (when all the arguments are serial) or in parallel (when at least one of the arguments is distributed).

[*] {cly,edelman,drcheng}@csail.mit.edu
[†] {gilbert,viral}@cs.ucsb.edu
[‡] Some of this text appears in Ron Choy's upcoming Ph.D. thesis

2 Functionality

Where possible, STAR-P leverages existing, established parallel numerical libraries to perform numerical computation. This idea is inherited from MATLAB*P. Several libraries already exist which provide a wide range of linear algebra and other routines, and it would be inefficient to reproduce them. Instead, STAR-P integrates them seamlessly for the user.

3 RT-STAP Benchmarks

The RT-STAP (Real-Time Space-Time Adaptive Processing) benchmark [1] is a benchmark for real-time signal processing systems developed by the MITRE Corporation. In the hard version of the benchmark which we run, the input to the MATLAB code is a data cube of 22 (channels) x 64 x 480 doubles. The code performs the following three steps:

1. Convert the input data to baseband.
2. Doppler processing.
3. Weight computation and application to find the range-Doppler matrix.

Upon execution, we noticed that step 1 was the most time consuming step. This is surprising, since the weight computation would be expected to have the highest FLOP count. It turns out that this is due to the MATLAB coding style used in the benchmark code. Since the point of the benchmark is to measure the running time of a typical application, we chose to proceed without modifying the code. The conversion step in the original MATLAB code is a *for* loop as follows:

```
for channum=1:NCHAN
  xx = CPI1_INITIAL(:,channum);
  CPI1(:,channum) = baseband_convert(xx, ...
                    SOME_ARGUMENTS);
end
```

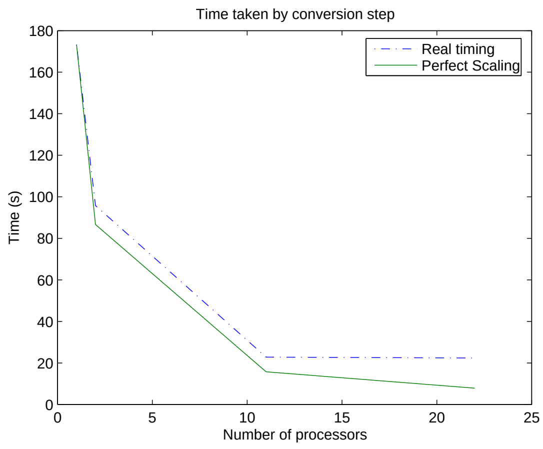

Figure 1: Scalability of RT_STAP

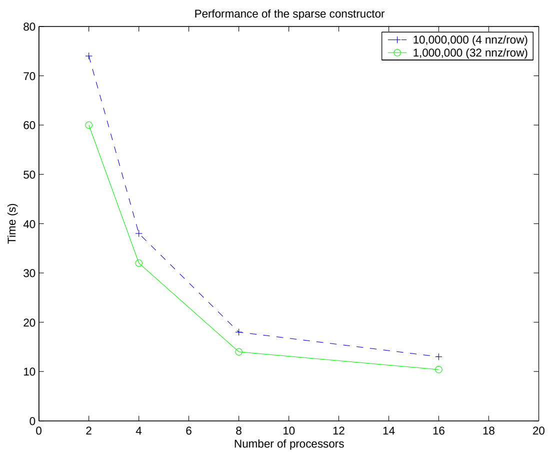

Figure 2: Scalability of *sparse* (SGI Altix 350)

It loops over the input channels and processes them in an embarrassingly parallel fashion. This makes it a natural candidate for STAR-P's multi-MATLAB mode. We converted the loop to run in STAR-P by changing the loop into a function call and putting it in an mm-mode call:

```
P_CPI1_INITIAL = matlab2pp(CPI1_INITIAL,2);
CPI1 = mm('convert_loop', SOME_ARGUMENTS);
CPI1 = CPI1(:,:);
```

Note that the calls before and after the mm call are used to transfer data to the server and back. The time required by these calls is also included in our timings.
Figure 1 compares timing results for sequential MATLAB and STAR-P on 2, 11 and 22 processors. The solid line shows the timings that would be obtained if the code scales perfectly. The real timings follow the solid line quite closely except for the 22 processors case. Going from 11 processors to 22 processors provides no additional benefits. This is easy to explain in terms of granularity. As the input data cube only has 22 channels, with 22 processors, each processor has only 1 channel of work, as opposed to 2 channels in the 11 processors case. So there is very little to gain from using additional processors, and any benefit is offset by the additional time needed for communication.

4 Sparse matrix capabilities

STAR-P provides basic sparse matrix capabilities [4] similar to those found in MATLAB. Sparse matrix algorithms are often useful in signal processing and embedded computing. Sparse matrix operations often display poor spatial and temporal locality resulting in irregular memory access patterns.

A very basic and fundamental sparse matrix operation is the sparse matrix constructor in MATLAB – *sparse*. It constructs a distributed sparse matrix from 3 vectors containing the row and column numbers and the corresponding non-zeros. The *sparse* constructor has fairly general applications in building and updating tables, histograms, and sparse data structures in general. It also accumulates and adds duplicate entries.

Figure 2 shows the performance of *sparse* on two matrices: one very large but sparse, with 10^7 rows and 4×10^7 non-zero entries; the other smaller and denser, with 10^6 rows and 32×10^6 non-zero entries.

References

[1] K. C. Cain, J. A. Torres, and R. T. Williams. RT_STAP: Real time space-time adaptive processing benchmark. Technical report, Feb 1997.

[2] R. Choy. Parallel Matlab survey. 2001. http://theory.lcs.mit.edu /∼cly/survey.html.

[3] P. Husbands and C. Isbell. MATLAB*P: A tool for interactive supercomputing. *SIAM PPSC*, 1999.

[4] V. Shah and J. R. Gilbert. Sparse Matrices in MATLAB*P: Design and Implementation. *Submitted to HiPC 2004*.