pMapper: Automatic Mapping of Parallel Matlab Programs*

Nadya Travinin, Henry Hoffmann, Robert Bond, Hector Chan, Jeremy Kepner, Edmund Wong
{nt, hoffmann, rbond, chanh, kepner, elwong}@ll.mt.edu
MIT Lincoln Laboratory, Lexington, MA 02420

Abstract

Algorithm implementation efficiency is key to delivering high-performance computing capabilities to demanding, high throughput signal and image processing applications and simulations. Significant progress has been made in compiler optimization of serial programs, but many applications require parallel processing, which brings with it the difficult task of determining efficient mappings of algorithms to multiprocessor computers. The pMapper infrastructure addresses the problem of performance optimization of multistage MATLAB® applications on parallel architectures. pMapper is an automatic performance tuning library written as a layer on top of pMatlab. pMatlab is a parallel Matlab toolbox that provides MATLAB users with global array semantics. While pMatlab abstracts the message-passing interface, the responsibility of generating maps for numerical arrays still falls on the user. A processor map for a numerical array is defined as an assignment of blocks of data to processing elements. Choosing the best mapping for a set of numerical arrays in a program is a nontrivial task that requires significant knowledge of programming languages, parallel computing, and processor architecture. pMapper automates the task of map generation. This abstract addresses the design details of the pMapper infrastructure and presents preliminary results.

Introduction

Automatic performance optimization of serial programs has been recognized as an important area of research and considerable progress has been made [1,2]. Additionally, efficient parallel algorithms exist for various functions such as FFT [3] and matrix multiplication [4]. The pMapper architecture is designed to tackle the global optimization problem of distributing signal and image processing applications that consist of multiple functions and computational stages.

The two primary goals of pMapper design are (1) *fast time to solution* and (2) *ease of programming*. pMatlab [5] allows MATLAB users not familiar with the message-passing style of programming [6, 7] to take advantage of parallel computers. However, mapping data objects is a non-trivial task requiring significant knowledge of parallel programming and parallel computer architecture. pMapper is designed for scientists not familiar with issues associated with parallel computing.

In the code in Figure 1, the arrays considered for distribution, A, B, C, D, and E (lines 2 and 3), are tagged with a special variable ***p***. This example illustrates that the changes (in bold italics) to the serial code are minimal.

In order to efficiently generate maps for arbitrary programs, pMapper requires an initialization phase. During initialization, pMapper collects information about the specifics of the parallel computer architecture and the pMatlab library. The collected information is used to construct a performance model. Once the performance model is constructed, it is used to generate maps for the tagged arrays. The mapping occurs at runtime and is done for each program submitted to the mapper. This naturally yields a two-phase system.

```
%Initialize variables
1. M...; N...; W...;
2. A=rand(M,N,p); B=zeros(M,N,p);
3. C=zeros(M,N,p); D=rand(M,N,p); E=zeros(M,N,p);
%Perform computation
4. B(:,:) = fft(A,[],1);        %FFT along columns
5. C(:,:) = fft(B,[],2);        %FFT along rows
6. E(:,:) = D*C;                %matrix multiply
7. E                            %print out E
```

Figure 1

pMapper Design Specifics

Phase 1: Initialization

Figure 2 provides a block diagram of the initialization phase of the mapping framework.

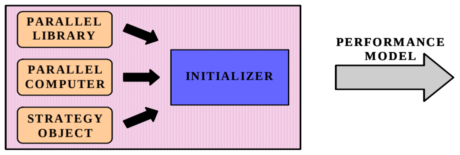

Figure 2

Initialization is done once at installation time. The inputs into the initializer are the parallel library information, the parallel computer specification, and the strategy object. The initializer runs timing experiments specific to the system and library version in order to construct the performance model. The strategy object is optional and contains general mapping heuristics provided by an expert parallel programmer.

The output of the initializer is the performance model used to generate maps. This model is used in the mapping and execution phase. The initializer and the resulting performance model differ with mapping approaches. For this iteration of the research the mapping was performed by a dynamic programming based approach, which used a timing database performance model. If, on the other hand, a neural network was used for the mapping and execution phase, then the performance model would be a trained neural network.

*This work is sponsored by the Department of the Air Force under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Phase 2: Mapping and Execution
pMapper uses lazy evaluation to collect as much information as possible about the program structure prior to assigning maps to distributed arrays. Another key design concept is runtime evaluation, which is made possible by the lazy evaluation of the program.

In the code example in Figure 1, lines 2 and 3, the output of an array constructor (`rand()`, `zeros()`, etc) produces a data object that stores necessary information about the array. No memory is allocated for the data of the numerical arrays at construction time. Similarly, when the `fft()` function call is made, no FFT is performed. Instead, tagged variables, along with functions that operate on them, are inserted into a signal flow graph. At this point in the program the variables and function calls exist in pMapper controlled space. The transfer of control from pMapper to the MATLAB environment occurs when the program requires access to the data. An example of such operation is the display operation, or simply omission of the semicolon in MATLAB syntax, as illustrated on line 7 in Figure 1. Intercepting MATLAB function calls and storing variables in pMapper space is possible through overloading of MATLAB functions, as was done in pMatlab [5] and Star-P [9].

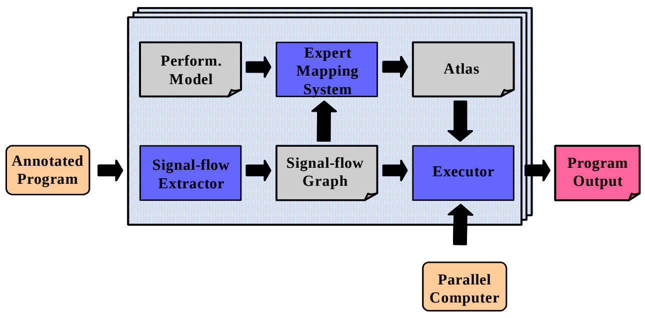

Figure 3

In Figure 3, the performance model is the output from the initialization phase (Figure 2). The Expert Mapping System (EMS) uses the performance model to generate the atlas, or collection of maps, for the program. Once the maps are generated, the executor combines the map information with the signal flow graph and executes the users' code.

Results

Figures 4 and 5 show both the speedup curve and the output maps for the application in Figure 1. This application has a high communication to computation ratio and is made up of important kernels present in many signal processing codes. The results were obtained using simulated timing data that described a low-latency architecture. While pMapper was originally designed for pMatlab running on a cluster, it also shows great promise for mapping applications to embedded systems.

Conclusion

pMapper is a two-phase mapping system designed as an automatic mapper for parallel MATLAB programs. It is written purely in MATLAB and currently is targeted as a mapper that produces pMatlab code for annotated MATLAB code. pMapper supports both multi-stage and multi-pipeline applications. The maps generated by pMapper allow for significant speedup of signal and image processing applications. The mapping overhead is low enough to allow MATLAB users to generate the mappings at runtime. Additionally, pMapper requires minimal changes to the code, thus making the task of writing parallel programs accessible to users unfamiliar with parallel programming issues. Initial experiments indicate that pMapper can also be used for other language implementations on other architectures.

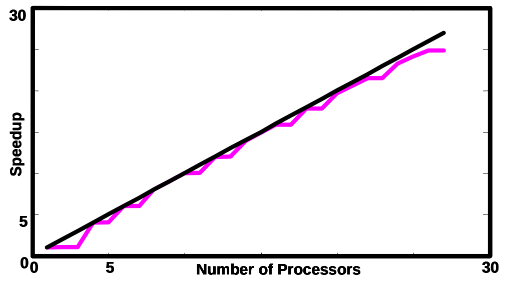

Figure 4

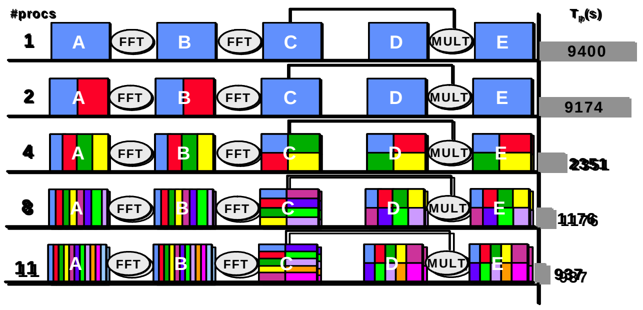

Figure 5

References

[1] A. Petitet, R.C. Whaley, J.J. Dongarra, "Automated Empirical Optimizations of Software and the ATLAS Project," *HPEC 2000 Workshop*, Lexington, MA, September 2000.

[2] J. Moura, M. Pueschel, M. Veloso, J.R. Johnson, R.W. Johnson, D. Padua, V. Prasanna, "SPIRAL: Automatic Implementation of Signal Processing Algorithms," *HPEC 2000 Workshop*, Lexington, MA, September 2000.

[3] M. Frigo and S.G. Johnson, "FFTW," http://www.fftw.org.

[4] Robert A. van de Geijn, *Using PLAPACK*. The MIT Press, 1997.

[5] Jeremy Kepner and Nadya Travinin, "Parallel Matlab: The next generation," *HPEC 2003 Workshop*, Lexington, MA, September 2000.

[6] J. Kepner, "Parallel Programming with MatlabMPI," *HPEC 2001 Workshop*, Lexington, MA, September 2000.

[7] J. Kepner, "300x Matlab," *HPEC 2002 Workshop*.

[8] Michael Wolfe, *High Performance Compilers for Parallel Computing*. Addison-Wesley, 1995.

[9] R. Choy, "Star-P: High Productivity Parallel Computing," *High Performance Embedded Computing (HPEC) Workshop 2004*, Lexington, MA, September 2004.