A Mapping Methodology for Designing Software Task Pipelines for Embedded Signal Processing [⋆]

Myungho Lee, Wenheng Liu, and Viktor K. Prasanna
Department of EE-Systems, EEB-200C
University of Southern California
Los Angeles, CA 90089-2562
http://ceng.usc.edu/~prasanna
{mlee + liu + prasanna}@halcyon.usc.edu

Abstract. In this paper, we present a methodology for mapping an Embedded Signal Processing (ESP) application onto HPC platforms such that the throughput performance is maximized. Previous approaches used a linear pipelined execution model which restrict the mapping choices. We show that the "optimal" solution obtained under that model can be improved, using the proposed execution model. Based on the new model, a three-step task mapping methodology is developed. The methodology is demonstrated by designing Software Task Pipelines for modern radar and sonar signal processing applications. Experimental results show improved performance using our approach over those obtained by previous approaches.

1 Introduction

In this paper, we address the problem of maximizing the throughput of an ESP application on a given number of processors of a High Performance Computing (HPC) platform. ESP applications are typically composed of a sequence of computation stages with varying computational complexities. Each stage consists of a number of identical tasks operating on disjoint sets of input data. The software developed for these applications are, in general, executed in a pipelined fashion. Thus, the resulting software is called a *Software Task Pipeline* (STP).

A task mapping methodology is needed to determine how the tasks of an application are mapped onto the processors of a target machine platform. An execution model which abstracts the execution of an application on a target machine is needed for task mapping. A linear pipelined execution model (linear model hereafter) has been widely used in the past [8, 9]. In [8], processor mapping algorithms using dynamic programming for optimizing the throughput is proposed, when a sequence of data parallel tasks is given. A stage is mapped onto a disjoint set of processors. Adjacent computation stages can be clustered and mapped onto the same set of processors. Stages can be also replicated such that a sequence of data sets can use replicated instances of a stage in a round robin

[⋆] Work supported in part by DARPA Embedded Systems Program under contract no. DABT63-95-C-0092 monitored by Fort Hauchuca.

fashion. However, the mapping choices allowed in this model are restricted. Their solution obtained under the linear model can be improved. Choudhary et. al. [2] also considered the problem of optimal processor mapping when a sequence of data sets is processed by a collection of tasks. The intertask data dependence of the tasks is assumed to form a series-parallel partial order. They also assume that each stage is mapped onto a disjoint set of processors. Their technique is also applicable to the problem considered in this paper. Our techniques can provide improved performance compared with their approach.

In this paper, we define a new execution model and develop a task mapping methodology for maximizing the throughput performance of an ESP application. The new model relaxes the restricted mapping choices allowed in the linear model. A new technique called *stage partitioning* is used to implement the new model. Based on the model and technique, a three-step task mapping methodology is proposed. The flexibility offered by our model leads to higher throughput performance than the claimed "optimal" solution [8] and also the solution proposed by [2].

The rest of the paper is organized as follows: Section 2 introduces our task model for ESP applications. A new execution model is proposed in Section 3. Using the new model, a mapping methodology is presented in Section 4. Experimental results for modern radar and sonar signal processing applications on IBM SP2 are shown in Section 5. Section 6 concludes the paper.

2 A Task Model

An ESP application consists of a sequence of computation stages. In each stage, a number of identical tasks are performed. Let n_i denote the number of tasks in Stage-i and t_i denote the time complexity of each task in Stage-i when such a task is performed on a single processor, where $1 \leq i \leq k$. The tasks in each stage execute on disjoint sets of input data. Each stage repeatedly receives its input data from the previous stage, performs computations, and sends its output data to the next stage. The first stage receives the external data (usually a 2-D or a 3-D data from the sensors) while the last stage produces the results (ex. Doppler bins, beam-patterns, etc.) as output. Figure 1 illustrates our task model representation for an adaptive Sonar beamformer.

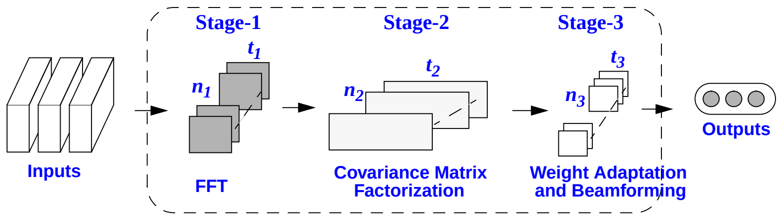


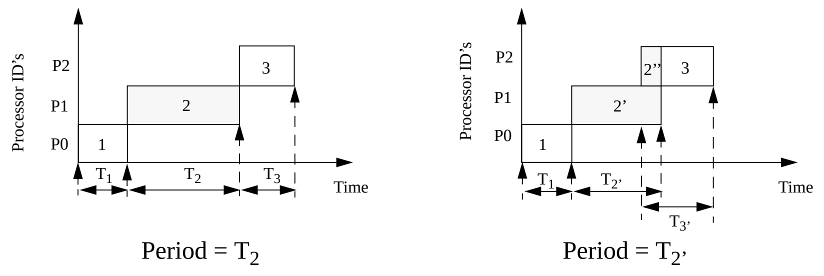
Fig. 1. Task model for an example application of embedded signal processing

The task models in [8, 2] assume the following: 1) All tasks in a stage (called task in their notation) can be executed on one or more processors. 2) The execution time of each stage is a function of the number of processors assigned to the stage. Even though they were motivated by signal and image processing applications, their formalism does not exploit scheduling of independent tasks in a stage.

3 A New Execution Model

In the linear model [8], the given number of processors is partitioned into a collection of disjoint sets. A set of processors receives its input data from the previous set, performs computations corresponding to a cluster of stages, and sends its output to the next set. Each processor, therefore, belongs to only one set. In the proposed model, this restriction is relaxed so that each processor may belong to a set as well as its adjacent sets. This allows more efficient and flexible task mapping. We will show later in Sections 4 and 5 that our model leads to higher throughput performance than the previous approach [8]. In [8], an optimal solution was obtained under the linear model.

In Figure 2, the linear model and our new execution model are illustrated using an example. Note that each task is mapped onto a single processor (and no more than a processor) according to our task model. Therefore, only task level, coarse grain parallelism is exploited. A task mapping assuming a linear model is shown in Figure 2 (a). The system throughput is determined by Stage-2, since it has the longest execution time (T_2). Using the proposed execution model, the throughput can be improved by partitioning the stage with the longest execution time. Stage-2 is partitioned into Stage-2$'$ and Stage-2$''$: some tasks in Stage-2 are associated with Stage-2$'$ and the remaining tasks are associated with Stage-2$''$. Later, when tasks are mapped onto processors, tasks in Stage-2$''$ are mapped onto the same set of processors as Stage-3 (see Figure 2 (b)). This results in a shorter period (thus, higher throughput). Stage partitioning is not considered in the previous approaches [8, 2]. In fact, their task models are not amenable to stage partitioning, whereas our task model facilitates stage partitioning.



(a) Using the linear model (b) Using the proposed model

Fig. 2. An illustrative example showing processor mappings using the linear model and the proposed model

4 A Mapping Methodology

Based on the execution model defined in the previous section, we propose a task mapping methodology for the design of software task pipelines. We consider the problem of maximizing the throughput of the resulting software task pipeline given a fixed number of processors. The methodology is described as follows:

- Step 1: **Data Remapping**
  In this step, the data layout of each stage is chosen based on the data access pattern of that stage. This determines the degree of task level parallelism of each computation stage (i.e., n_i for Stage-i). Data remapping techniques are used to change the data layout between successive stages so that each processor contains all the data needed to execute a task. Data remapping is a critical step in the mapping process. (See [5] for an example that illustrates the impact of data remapping in parallelizing Higher-Order post-Doppler (HOPD) STAP.)
- Step 2: **Coarse Resource Allocation**
  The available processors are initially allocated to the stages by estimating the time complexity of each computation stage. A task in each computation stage is executed on a single processor and hence, its execution time (i.e., t_i for each task in Stage-i) can be estimated. Then the number of processors allocated to Stage-i is determined in proportion to the time complexity of that stage (i.e., $n_i \times t_i$). If P_i processors are allocated to Stage-i and $P_i > n_i$, then Stage-i is replicated $\lceil \frac{P_i}{n_i} \rceil$ times. The P_i processors allocated to Stage-i are evenly distributed to these replicated stages. This guarantees that sufficient degree of task level parallelism exists in each pipeline stage.
- Step 3: **Fine Performance Tuning**
  Based on our execution model, the computation stages can be partitioned to improve the throughput of the pipeline generated in Step 2. Stage partitioning is realized by using a heuristic algorithm. This heuristic reduces the period of the pipeline in an iterative manner. In each iteration, a bottleneck stage (a stage with the longest execution time) is identified. Let this stage be Stage-i. Based on our proposed execution model, the algorithm locally minimizes the period by 1) re-assigning the processors allocated to stage-i and Stage-(i+1) (Stage-(i-1)). 2) redistributing the tasks in Stage-i and Stage-(i+1) (Stage-(i-1)) to minimize the period. A new task mapping which results in the largest reduction of the period is chosen. This step is repeated until the period cannot be further reduced.

Using Step 2 and Step 3, the tasks of an ESP application are mapped onto the target HPC platform. Then the data remapping algorithm specified in Step 1 is used to perform the inter-stage communication.

The mapping methodology and its effectiveness are first illustrated by an example. For the sake of illustration, we assume the communication cost between the adjacent stages to be zero. Note that, in Step 1, the communication between stages is explicitly performed by remapping the data. In this example, a two-stage application is mapped onto 6 processors. Let $n_1 = 5, n_2 = 7$, $t_1 = 1$ second and $t_2 = 1$ second. Based on the execution model in [2], the best throughput that can be achieved is 0.33/sec (as shown in Figure 3 (a)). Two optimal mappings using stage clustering and replication are shown in Figures 3 (b) and 3 (c). However, clustering or replication cannot improve the throughput. When our mapping methodology is used, the throughput can be significantly improved. The coarse resource allocation results in an initial mapping such that three processors are allocated to each stage. Then fine performance tuning is performed. As shown in Figure 3 (d), Stage 2 is partitioned. Five of the tasks in Stage 2 are clustered with the tasks in Stage 1. The resulting task cluster is mapped onto five processors. The remaining two tasks in Stage 2 are then mapped onto the last processor. This mapping can fully utilize the computing power. Using our mapping methodology, the throughput is increased to 0.5/sec. Note that, in this example, the throughput obtained by our mapping is optimal.

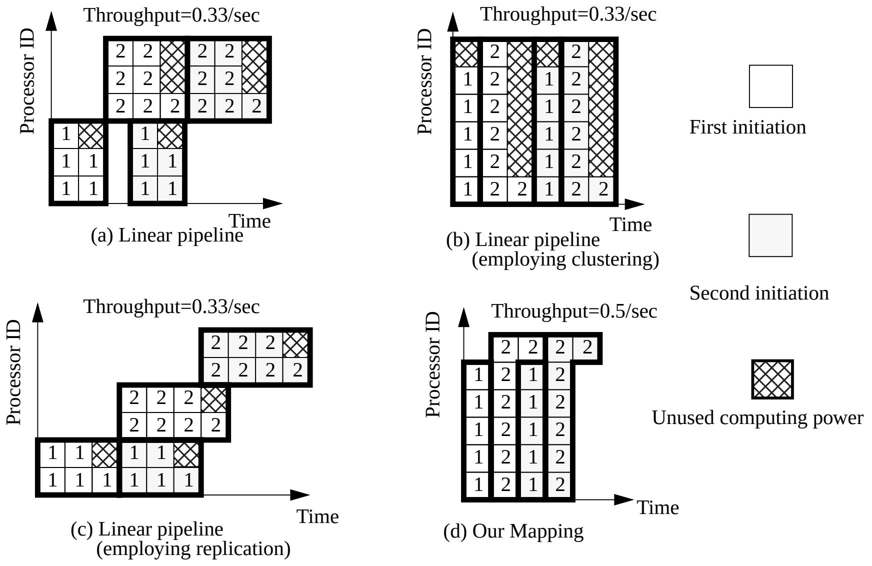


Fig. 3. An illustration of our mapping methodology and its comparison with linear model (with clustering and replication)

5 Experimental Results

We developed STP's for First-Order Doppler-Factored STAP benchmark [3] and three-stage adaptive Sonar beamforming benchmark [4] on an IBM SP-2 to show the effectiveness of our mapping methodology. To compare our mapping methodology which is denoted as $A3$, we also developed two designs based on the previous approaches, denoted as $A1$ and $A2$. These two approaches examine all possible task mappings under the linear model used in [8]. However, no clustering or replication is used in $A1$. $A2$ also uses the linear model but allows clustering and replication of pipeline stages. Note that $A1$ and $A2$ are the best performance that can be achieved using techniques in [2] and [8] respectively. It is not stated in [8] whether data remapping is explicitly performed or not. Thus, $A1$ (as well as $A2$) may use a fixed data layout throughout the computation stages. Since data remapping can significantly improve performance, the performance improvement using $A2$ can be due to the use of data remapping. To expose performance improvement using our task model and stage partitioning, we also perform data remapping between adjacent stages (Step 1) for both $A1$ and $A2$. Therefore, in the following, the observed performance improvement can be attributed to various task mapping techniques.

5.1 Results for Sonar Benchmark

This beamformer uses 64 sensor elements to sample the acoustic signals. 64 frequency bins are chosen with 128 beams formed in each frequency bin. The details of the beamformer can be found in [4]. Based on the experiments performed on SP-2 and the features of the benchmark, the time complexity of each task and the number of tasks in each stage are shown in Table 1.

Table 1: Characteristics of the MVDR sonar beamformer

Processing stage	Functionality	Time complexity of each task	Number of tasks
Stage 1	FFT	88 μsec	64
Stage 2	Covariance matrix factorization *	68,500 μsec	64
Stage 3	Weight adaptation and beamforming	1900 μsec	8192

* Using the LAPACK subroutine, **cgesvd**, to perform single-precision complex-number singular value decomposition (SVD) on SP-2.

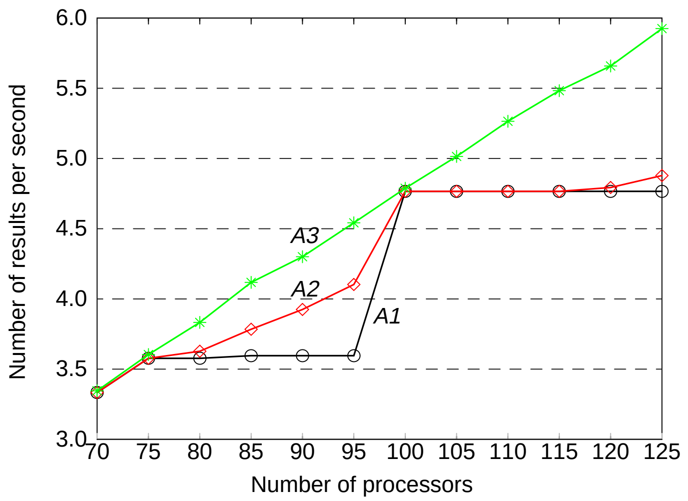


Fig. 4. Performance Comparison of $A1$ through $A3$

Figure 4 shows the throughput performance on an IBM SP-2. The number of processors in this experiment is varied from 70 to 125 (in increments of 5 processors). In this experiment, a practical latency constraint of 1 second is assumed to meet the real-time processing requirement. The results show superior performance of our design over $A1$ and $A2$. For instance, our approach increases the throughput by approximately 24.3% compared with $A1$ when 125 processors are used. This figure also shows that our approach increases the throughput by approximately 21.4% compared with $A2$ when 125 processors are used. Besides, Figure 4 illustrates significant improvement in scalability using our methodology. The sub-linear speedups using $A1$ and $A2$ are mainly due to the restrictions imposed on their corresponding execution models.

5.2 Results for Radar Benchmark

A First-Order Doppler-Factored STAP described in [3] without preprocessing step was implemented. A data cube size of 16 (number of channels) $\times$ 64 (PRIs) $\times$ 480 (rangegates) is assumed. There are three processing stages in this application: Stage 1: Doppler processing; Stage 2: Weight computation by covariance matrix factorization; Stage 3: Weight application. The time complexity of each task running on one node of IBM SP-2 and the number of tasks in each stage are shown in Table 2. Library routines (IBM essl library and LAPACK) were used for implementing the computational kernels involved in each stage.

Table 2: Characteristics of First-Order Doppler-Factored STAP

Processing Stages	Computations Involved	Time complexity of each task	Number of tasks
Stage 1	Doppler Processing	28 μsec	7680
Stage 2	Weight Computation	9,800 μsec	384
Stage 3	Weight Application	100 μsec	384

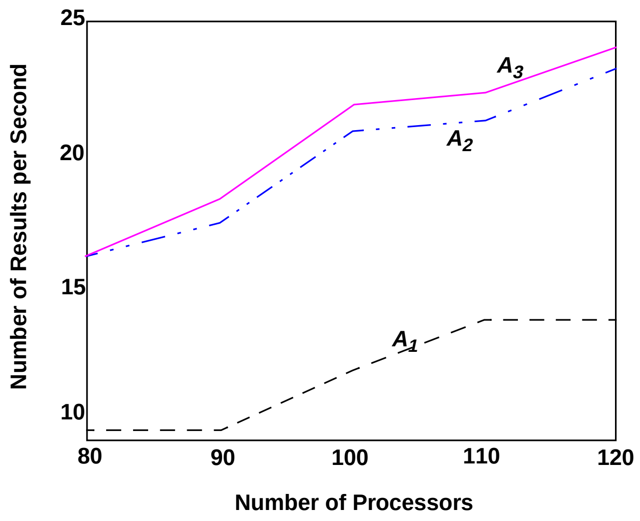


Fig. 5. Performance Comparison of $A1$ through $A3$

Figure 5 shows the throughput performance of A_1 through A_3 on IBM SP-

2. The number of processors was varied from 80 to 120 (in increments of 10 processors). A practical latency constraint of 0.3 seconds is assumed to meet the real-time processing requirement. The results show superior performance improvement of our design over $A1$ and some improvement over $A2$. For instance, our approach increases the throughput by approximately 85.5% compared with $A1$ and by 4.4% improvement A_2 when 100 processors are used.

6 Concluding Remarks

This paper presented a task mapping methodology for ESP applications. Previous task mapping approaches do not effectively utilize the available processors due to the restricted linear model. Therefore, the resulting task mapping (which was claimed to be optimal) can be improved. We defined a task model to represent an ESP application and developed a new execution model. Based on the new model, we proposed a task mapping methodology. Software task pipelines designed for radar and sonar signal processing benchmarks using our mapping methodology show that the proposed methodology leads to significant performance improvement over the previous approaches.

References

1. R. Bernecky, "Sonar Beamforming Challenge Problems," Presented at the DARPA/ITO Embeddable Systems PI Meeting, San Diego, June 1996.
2. A. Choudhary, B. Narahari, D. Nicol, and R. Simha, "Optimal Processor Assignment for a Class of Pipelined Computations," *IEEE Trans. Parallel and Distributed Systems,* Vol. 5, No. 4, April 1994, pp. 439-445.
3. R.A. Games, J.A. Torres, and R.T. Williams, "RT_STAP: Real-Time Space-Time Adaptive Processing Benchmark", MITRE Corporation, June 1996.
4. M. Leonhardt, "Implementation of Minimum Variance Distortionless Response (MVDR) Adaptive Beamforming Algorithm," NUSC Technical Document 8453, July 1989.
5. Y. W. Lim and V. K. Prasanna, "Scalable Portable Implementations of Space-Time Adaptive Processing", 10th International Conference on High Performance Computers, Ottawa, Canada, 1996.
6. Y. W. Lim, P. B. Bhat, and V. K. Prasanna, "Efficient Algorithms for Block-Cyclic Redistribution of Arrays," IEEE Symposium on Parallel and Distributed Processing, Oct. 1996.
7. W. Liu and V. K. Prasanna, "Software Pipelines for Real-Time Adaptive Sonar Beamforming," Manuscript, Department of EE-Systems, University of Southern California, June 1997.
8. J. Subhlok, and G. Vondran, "Optimal Mapping of Sequences of Data Parallel tasks," in proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, July 1995.
9. J. Subhlok and G. Vondran, "Optimal Latency-Throughput Tradeoffs for Data Parallel Pipelines," in proceedings of Eighth Annual ACM Symposium on Parallel Algorithms and Architecture (SPAA), June 1996.
10. C. L. Wang, P. B. Bhat, and V. K. Prasanna, "High-Performance Computing for Vision," in proceedings of the IEEE, vol. 84, No. 7, July 1996.

This article was processed using the LaTeX macro package with LLNCS style