The PCP/PFP Programming Models on the BBN TC2000*

Eugene D. Brooks III, Brent C. Gorda, Karen H. Warren
Massively Parallel Computing Initiative
Lawrence Livermore National Laboratory
Livermore, California 94550

Abstract: We describe the PCP/PFP programming models which we are using on the BBN TC2000. The parallel programming models are implemented in a portable manner and will be useful on the scalable shared memory machines we expect to see in the future. We then describe the TC2000 machine architecture which is a scalable general purpose parallel architecture capable of efficiently supporting both shared memory and message passing programming paradigms. We also briefly describe a PCP implementation of the Gauss elimination algorithm which exploits the large local memories on the TC2000.

1 Introduction

Microprocessors have made incredible strides in performance in recent years and are beginning to overrun traditional supercomputer performance for scalar dominated application codes. It is anticipated that supercomputer class vector processing performance will appear in microprocessor form in the next few years. This development is enabling a new breed of supercomputers composed of hundreds, and in some cases thousands, of high performance microprocessors.

The BBN TC2000 is a scalable microprocessor based machine which provides a shared memory facility through a multi-staged interconnection network. It is similar to the IBM RP3 architecture [1] but is currently available commercially. Because the machine supports both high bandwidth interleaved shared memory and large local memories, it is well suited to supporting both shared memory and message passing programming models.

We have implemented FORTRAN and C versions of the split-join [2] parallel programming paradigm on the BBN TC2000 with the Parallel C Preprocessor (PCP) and the Parallel FORTRAN Preprocessor (PFP).

The split-join parallel programming model is highly portable because a full featured version is easily implemented with a preprocessor and relatively little back end compiler support. An earlier version of PCP has been used on a variety of machines, including Sequent Symmetry, Sequent Balance, Alliant FX/8, SGI, Stellar, and Cray multiprocessors. PFP was written specifically for the large base of FORTRAN users who are participating

---

*Work performed under the auspices of the U. S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.
in the Massively Parallel Computing Initiative at Lawrence Livermore National Laboratory. The PCP and PFP preprocessors have an option of generating efficient serial code, and this has been used to target both multiprocessors and uniprocessors with the same source code. We have found the split-join programming model to be a good match to the BBN TC2000 architecture. Users have found that one must exploit nested concurrency effectively in order to use successfully large numbers of processors on general purpose applications. The split-join programming model implemented in PCP and PFP provides the efficient means for exploiting nested concurrency effectively.

The sections of this paper are as follows. The split-join model and its memory model are described in Sections 2 and 3. Specifics on how the implementation of these models take advantage of the architecture are included. The synchronization primitives offered in PCP and PFP are discussed in Section 4. The actual PCP syntax is shown in Section 5. The BBN TC2000 hardware and capabilities are presented in Section 6. A PCP implementation of the Gauss Elimination algorithm which exploits the large local memories is discussed in Section 7.

2 The Split-join Model

In the traditional fork-join parallel programming model, a single processor starts the execution of the program and acquires more processors as concurrency is encountered in the code. Examples of this model are PCF [5] in which the system may utilize more processors for designated program sections if it desires, and the Cray autotasking model [6] which automatically distributes loop iterations to a number of multiple processors determined by the system and which itself obviously is not portable. The acquisition of more processors is generally an expensive operation. Nonetheless the fork-join programming model has been quite useful on tightly coupled shared memory machines with relatively few processors, and on some architectures such as the Alliant FX/8 and the Convex C2 which provide special hardware to make the dispatch of slave processors occur as quickly as possible. Scalable machine architectures are not as tightly coupled and the cost of communication between processors, heavily used in the process of dispatching processors in the fork-join model, is relatively high.

In the split-join paradigm we deal with the high cost of processor dispatch and the high cost of communication between processors by minimizing their occurrence in the fundamental constructs of the programming model. All of the processors the job will ever acquire are dispatched at the start of the program and are immediately placed under the control of the programmer. This set of processors which loosely follow each other through the code is referred to as a team of processors. At this level the split-join parallel programming model is very similar to Harry Jordan's Force [3] and the IBM SPMD [4] programming model, the most significant differences being the support for team splitting, the determinism which the user can establish with respect to which processors do what, and the arbitrary nesting of concurrency constructs.

The user is in complete control of the initial team of processors. Nested concurrency is exploited by the user through the notion of team splitting. In the static form of team splitting, the user explicitly marks off separate blocks of work which can be executed independently of each other (it is assumed that each block of work is itself a job consisting...