SOFTWARE DEVELOPMENT ISSUES FOR PARALLEL PROCESSING

Lucian Russell and R.N.C. Lightfoot

Computer Sciences Corporation

ABSTRACT

Parallel architectures for computers give rise to several issues with respect to software development. The issue to be resolved depends on the life-cycle phase in which the problem appears. During problem analysis and program design, the issue is best resolved by using an intelligent language environment, exemplified by the Rational R1000 Ada machine. During the post-implementation phase, an existing program can be restructured by using an expert tool that generates directed graph models of the program, and then analyses the models and the source to identify potentially parallelizable code sequences, and through workstation window oriented displays guides the user through the process of restructuring a program to efficiently run on a parallel architecture machine.

As parallel architectures in computing hardware become more common, there exists an increasing need for software that can be allocated to parallel computational elements and, therefore, take maximum advantage of the hardware features.

Historically, the major portion of programmers' effort in creating scientific and engineering (S/E) programs has been focused on optimizing the timing and storage efficiency of the solution algorithms on single-processor systems. However, an effort to make the best use of parallel architectures must go far beyond programmers' manual attempts to restructure existing S/E programs to enable them to run efficiently on the new machine architectures. In fact, the new parallel hardware architectures give rise to several software development issues across the various life-cycle phases; analysis, design, implementation, and the incremental re-development of programs, the activity usually called program maintenance.

One of the most obvious improvements would be a toolset that could offer intelligent program restructuring suggestions to the programmer, and thereby eliminate his tedious, time-consuming, highly error-prone, manual change procedure. The following figure identifies the various phase-related software improvements which could substantially reduce the labor to create resource-efficient programs for parallel architectures.

<table>
<thead>
<tr>
<th>ANALYSIS</th>
<th>DESIGN</th>
<th>IMPLEMENTATION</th>
<th>MAINTENANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPROVED RUN TIME ENVIRONMENT SUPPORT BY HARDWARE VERSIONS OF MATHEMATICAL TRANSCENDENTAL FUNCTIONS AND FOR MATRIX OPERATIONS AND/OR PATTERN RECOGNITION ALGORITHMS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXTENDING AVAILABLE FUNCTIONALITY BY ADDING COMMONLY USED FUNCTIONS TO STANDARD SOFTWARE LIBRARY SET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LANGUAGE- AND HARDWARE-SENSITIVE GRAPHICALLY-ORIENTED EDITORS AT USER WORKSTATIONS THAT SUPPORT SCOPE AND EXTENT ANALYSES, AND THAT RECOGNIZE AND IDENTIFY TO THE PROGRAMMER SEQUENCES OF PARALLELIZABLE CODE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXPERT SYSTEM TOOL TO CAPTURE COMPILED PARSING INFORMATION AND TO CREATE MULTIPLE DIRECTED-TREE MODELS, INCLUDING A PETRI-NET MODEL, AND TO SIMULATE THE PROGRAM IN ITS PETRI-NET FORM, AND USE THE P-N SIMULATION RESULTS TO PREDICT THE USAGE TIMES OF EACH INDIVIDUAL PROCESSOR ELEMENT, THEN CREATE A RUN-TIME EXECUTION TRACE THAT COMPARES ACTUAL VERSUS PREDICTED USAGE TIMES; AND TO USE ALL KNOWLEDGE COLLECTED TO MAKE RECOMMENDATIONS TO THE USER/PROGRAMMER REGARDING THE BEST CHANGES TO BE MADE TO THE ORIGINAL SOURCE PROGRAM TO OPTIMIZE THE RESOURCE UTILIZATIONS.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The issue of improved run-time support, in the framework of parallel architectures, can be resolved; (1) by eliminating the centralized repository of software routines that supply standard mathematical functions and operations and by substituting, at each processor in a parallel configuration, hardware chip versions of those functions; or (2) hand-coding by experts of those functions that are part of the standard language definition to make programming for efficiency easier.

The issue of improved program analysis, design, and implementation could be resolved by graphically-oriented editors that are completely language-sensitive, as well as hardware sensitive, which provide scope and extent analysis and that will recognize and identify, to the programmer, all sequences of source code that are eligible for parallel allocation.

The resolution of the software development issues in the analysis, design, and implementation phases, discussed above, can be based on the technology of existing systems. The issue of improved support for program maintenance, or incremental redevelopment of programs, cannot be resolved quite so simply, or so straightforwardly.

In the various, large, scientific and engineering computation facilities around the world there exists many millions of lines of FORTRAN code that have been generated to execute on single processors, or on closely-coupled multi-processor systems. Very many of those existing programs are very large, and cannot be easily divided into discrete and separate functional components that could be allocated to separate processors, and thereby, take advantage of the computational parallelism offered by the new architectures. What is needed is an expert system tool or toolset that can support the programmer in making the data analyses, and in recognizing parallelizable code sequences.

Our own, on-going research into this problem is based on transforming the parsed source code into control flow graphs, data flow graphs, and into colored, timed Petri-Nets. The computer sciences research community has demonstrated the isomorphisms of these three types of directed multigraphs; nonetheless, we have elected to generate and to use all three models because each one offers a particular type of insight into the analysis of systems that is not offered by the other two models. The Petri-Net (P-N) model, however, forms the basis of our analyses for the simple reason that there exists a much larger body of tools for analyzing P-Ns than for the other two types of graphs. The P-N reachability graph analysis tools that are available in our software engineering laboratory enable a ready identification of the different program states and the selection of source code sequences that can be allocated to separate processors.

We are developing an 8-step methodology for reworking existing FORTRAN programs for more efficient execution on parallel architecture machines.

Our methodology is summarized in the following:

STEP 1 PARSE FORTRAN SOURCE CODE, CREATING A PARSE TREE AND SYMBOL TABLE. ADD TO THE KNOWLEDGE BASE (KB).

STEP 2 CREATE CONTROL FLOW GRAPHS AND DATA FLOW GRAPHS USING KB. ADD CONTROL FLOW AND DATA FLOW GRAPHS TO KB.

STEP 3 CREATE EQUIVALENT COLORED, TIMED PETRI-NET (P-N) USING KB. ADD THE GENERATED NET TO KB.

STEP 4 EXTRACT FROM KB THE DENOTATION OF THE TARGETED COMPUTER. IDENTIFY PROGRAM SEQUENCES (PSEQ) ALLOCABLE TO SEPARATE PROCESSORS. MAP PSEQs TO CPUs. ADD UPDATED P-N AND OTHER MODELS TO REFLECT THE CHANGED PROGRAM.

STEP 5 COMPILE RESTRUCTURED SOURCE PROGRAM; LOAD AND LINK. ADD OBJECT CODE TO KB.

STEP 6 CONDUCT SIMULATION OF PETRI-NET. PREDICT EXPECTED CPU USAGE. ADD RESULTS OF SIMULATION AND PREDICTED USAGE TIMES TO KB.

STEP 7 MONITOR RUNNING FORTRAN PROGRAM WITH THE INSTRUMENTED RUN-TIME ENVIRONMENT. RECORD EXECUTION TRACES. GENERATE TIME-LINE GRAPH FOR EACH CPU SHOWING ACTUAL USAGE TIMES. CREATE COMPOSITE GRAPH EXHIBITING THE DELTA BETWEEN PREDICTED AND ACTUAL TIMES FOR EACH CPU. ADD ALL INFORMATION TO KB.

STEP 8 USING KB, GENERATE SET OF WINDOW-ORIENTED DISPLAYS THAT SUPPORT USER/PROGRAMMER IN PROGRAM RE-RUN SCENARIOS, AND HIS ATTEMPTS TO OPTIMALLY ALLOCATE CODE TO CPUs.