International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

Volume 5 Issue 2 July 2015

International Manuscript ID : ISSN22308849-V5I2M89-072015

PREDICAMENTS IN AUTOMATIC PARALLELIZATION OF COMPUTATIONALLY EXPENSIVE APPLICATIONS

Nahar Priyank Hasmukhbhai
Research Scholar
Shri Venkateshwara University
Uttar Pradesh, India

Dr. Parveen Kumar
Professor
Shri Venkateshwara University
Uttar Pradesh, India

ABSTRACT

For development of parallelizing compilers application test suites used are either single file based programs or algorithm structures. The full scale applications primarily commercial pose many challenges which are seldom addressed. Thus for these applications automatic parallelization is considered sometimes to be inconceivable. This paper tries to surface the impediments to be addressed so that the parallelization techniques may be applied to these commercial programs. A benchmark suite specifically designed to exhibit the computing needs found in industry has been used. Benchmarks are from High Performance Group of the Standard Performance Evaluation Corporation SPEC. Parallel and serial versions of both applications are available. The parallel variants are hand parallelized with shared memory directives either at the largest level of granularity or in a hybrid manner where MPI is used at the largest level of granularity and OpenMP directives are used at a lower level.

In this paper the parallel variants are compared with the automatically parallelized serial codes. Polaris parallelizing compiler is employed which takes language (formula based) codes and inserts OpenMP directives around loops determined to be dependence free. Various challenges faced by an automatic parallelizing compiler were found when dealing with full applications modularity, legacy optimizations symbolic analysis, array variations and issues arising from input output operations. The results presented in

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

Volume 5 Issue 2 July 2015

International Manuscript ID : ISSN22308849-V5I2M89-072015

this paper shall benefit parallelizing compilers with capabilities for handling large scale science and engineering applications

Keywords: Parallelization, Computational Applications, Compiler Techniques

1. INTRODUCTION

Any programming language, compiler, operating system and system architecture will ultimately have to improve upon its functionality and performance for applications that have commercial existence. Commercial applications are generally voluminous in terms of line of codes and data sets are widely used and are usually not freely available. Most programs that are being used to drive and evaluate the design of new computer systems technology do not fit this definition of commercial applications. Systems research typically uses benchmarks that have reasonably short execution times and are publicly available. Short runtimes are important because, it is not unusual that in the course of a research project a test program is run many times. If architecture simulators are used, these programs run two to three orders of magnitude slower than on an ordinary computer. Public availability of test programs is essential for all scientific research because research, results are of small value if cannot be reproduced by other research groups.

The long term goal of the research project described in this paper is to advance automatic parallelization technology for high performance computers. A test application for such research typically includes suites such as the SPEC CPU, Perfect, or Linpack benchmarks. In this paper, study two programs that come close to definition of commercial applications is been done. Here two applications are used from the SPEChpc benchmark suite, called SPECseis and SPECchem. Both codes are large scale computational applications that reflect problems faced in commercial settings. Applications are being used that are commercially relevant while still obtaining results that can be reproduced and shared publicly.

SPECseis [5] was developed by ARCO beginning in 1993 to gain an accurate measure of the performance of computing systems as it relates to the seismic processing industry for procurement of new computing resources. The current SPECseis is missing Kirko and pre stack migration techniques.

Other application package, SPECchem [4] is used to simulate molecules ab initio, at the quantum level. It is a current research effort under the name of GAMESS at the Gordon Research Group of Iowa State University and is of interest to the pharmaceutical industry. Like SPECseis, SPECchem is often used to exhibit performance of high performance systems among the computer vendors.

The contribution of this paper is to show program patterns of commercially relevant HPC applications that pose significant problems for automatic parallelization. Both SPECseis and SPECchem are parallelized using OpenMP. For this study, commented out the OpenMP directives and used the manual parallelization as standard for evaluating how well parallelizing compiler performs. Then, reasons are analyzed that why a parallelizing compiler could not detect the same level of parallelism. The compiler used is the Polaris translator [1], one of the most advanced parallelizing compilers to date. Section 2, describes five categories of challenges

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

Volume 5 Issue 2 July 2015

International Manuscript ID : ISSN22308849-V5I2M89-072015

faced by parallelizing compilers. Sections 2.1 describe issues arising from the fact that large applications naturally have a very modular structure. Section 2.2 shows examples of "Endowment Accumulation" that compilers must recognize. Section 2.3 discusses the need for advanced symbolic analysis, Section 2.4 deals with the issue of array variations at subroutine boundaries; and Section 2.5 describes problems in the presence of input output operations. Section 3 concludes the paper.

2. IMPEDIMENTS OF AUTOMATIC PARALLELIZING COMPILERS

Using compiler tools on large-scale applications it is found that success rate is significantly less. Here present code examples that illustrate these challenges are described below and discuss possible improvements to compiler technology. The regular access patterns representative of computational codes can be extracted from a full application suite of codes, automatic parallelizing compilers and parallelization techniques.

2.1. Modularity

Large-scale applications naturally have a tendency to be structured into many modules. Modularity is a general software engineering tool. Moreover, library modules may be included that perform some of the desired functionality. Full applications have deep levels of hierarchy that include abstractions with interfaces to the different computational routines. Modular programs generally raise the compiler issue of inter procedural analysis. In this work, issue has become significantly important. To obscure this issue, it is not always known at compile time which of the functions will be called during a specific execution.

```
Do jproc = jtop, nproc
 IF (name .EQ.'DCON')
  THEN
IF (X . EQ . 'A')
  THEN
  CALL
DCONA(ldim,maxtrc,otr,nra,ra,nsa,sa,abort,ipr)
 ELSE IF (x .EQ. 'B')
  THEN
  CALL
DCONB(ldim,maxtrc,otr,nra,ra,nsa,sa,abort,ipr)
 ELSE IF ( name. EQ. 'DGEN')
  THEN
 ELSE IF (name.EQ. 'DMOC')
  THEN
 ENDDO
```

Figure 1. Driver Routine, SEISPROC, from SPECseis.

2.1.1. Functions of Dynamic Application

Both of applications include a large body of functionality. Only a small part of the code is used in any specific execution. For example SPECseis typically runs in four "phases", called data generation, data stacking, depth migration and time migration. The specific routines invoked are determined by the input data. Because of this, compiler could not determine at compile time what routines would be called.

The code example in Figure 1 shows how a driver routine is used to implement this form of dynamic subroutine invocation in SPECseis. The variable name is derived from input data. Accordingly, the compiler

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

Volume 5 Issue 2 July 2015

International Manuscript ID : ISSN22308849-V5I2M89-072015

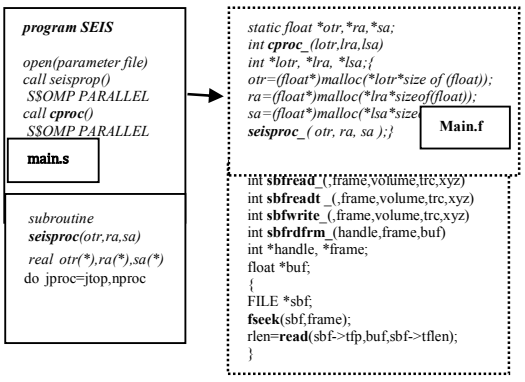
cannot determine which routines are called or in which order are called and concludes that there are cross iteration dependencies within this region.

To triumph over the problem of not knowing which routines will be called at compile time would require program knowledge. With SPECseis, this would mean that the compiler must know that the seismic routines are only applied to the seismic traces in certain orders. The compiler would also have to understand that the data originates from only few locations in the code. With this knowledge and with extensive expression propagation an automated compiler may be able to find parallelism encompassing a driver routine.

2.1.2. LINGUAL HURDLES

Another result of code modularity is multilingual applications; SPECseis has a Fortran 77 main program, which calls a C routine, to allocate memory. The code sections in Figure 2 illustrate these situations.

As new languages become widely used and compilation techniques for higher level languages of the object oriented flavor are developed to produce efficient code and expected to see the instances where the optimizing compiler must cross language barriers within a single application to grow with time. To overcome this hurdle, the compiler must perform interprocedural analysis across languages.



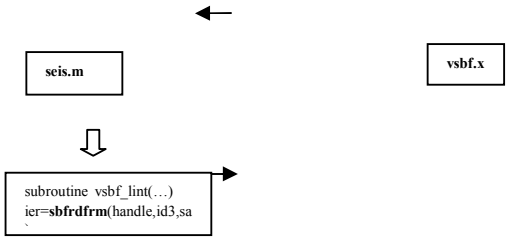


Figure 2. Multi-Lingual Characteristics of SPECseis

2.1.3. Extensible Libraries

These applications make use of software libraries. One characteristic of these library routines is that it tends to have many options and parameters. Figure 3 shows an example from SPECseis.

The library routine SCOPY is used to copy one vector into another with any stride for either of the two vectors. Similar examples could be given from SPECchem, such as the DDOT routine shown later in Figure 4. DDOT is almost always called with the strides of the two vectors (incx and incy) equal to 1.

2.2. Endowment Accumulation

SPECseis and SPECchem both use endowment code for low level mathematical functionality. SPECseis includes 35 IEEE library routines to perform Fast Fourier Transformations. SPECchem includes 63 matrix routines, some of which were derived from Linpack code of 1978. These codes tend to be optimized for performance, but may hinder additional compiler optimizations. For example, the DDOT routine in SPECchem simply produces the dot product of two vectors. The simple code for the general case is shown on the left side of Figure 4.On the right, a form

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

Volume 5 Issue 2 July 2015

International Manuscript ID : ISSN22308849-V5I2M89-072015

of DDOT is shown that was transformed for improved locality of data references.

```
SUBROUTINE SCOPY (n, a, inca, b, incb)
Copy a vector with stride, BLAS version
    INTEGER n, inca, incb, i, ia, ib
    REAL a (*), b (*)
If stride is negative, start from end of vector
    IF (inca .LT. 0) THEN
        ia = 1 + inca*(1-n)
    ELSE
        ia = 1
    ENDIF
    IF (incb .LT. 0) THEN
        ib = 1 + incb*(1-n)
    ELSE
        ib = 1
    ENDIF
Loop and copy from a to b
    DO i = 1, n
        b(ib) = a(ia)
        ia = ia + inca
        ib = ib + incb
    ENDDO
    RETURN
    END
```

Figure 3. Library Subroutine SCOPY of SPECseis.

```
DOUBLE PRECISION FUNCTION
DDOT(n,dx,* incx,dy,incy)
DOUBLE PRECISION dx(*),dy(*)
    DDOT = 0.0D+00
    dtemp = 0.0D+00
    ix = 1
    iy = 1
IF(incx .LT. 0) ix = (-n+1)*incx + 1
IF(incy .LT. 0) iy = (-n+1)*incy + 1
    DO 10 i = 1,n
    dtemp = dtemp + dx(ix)*dy(iy)
    ix = ix + incx
    10 iy = iy + incy
    DDOT = dtemp
RETURN
END
 USES UNROLLED LOOPS FOR
INCREMENTS EQUAL TO ONE.
C JACK DONGARRA, LINPACK, 3/11/78.
    20 m = MOD (n,4)
IF (m .EQ. 0) GO TO 40
    DO 30 i = 1,m
    30 dtemp = dtemp + dx (I)*dy(I)
    IF( n .LT. 4 ) GO TO 60
    40 mp1 = m + 1
    dtloc = 0.0D+00
DO 50 i = mp1, n, 4
    50 dtloc = dtloc + dx(i)*dy(i) + dx(i +
        1)*dy(i + 1)
    +* dx(i + 2)*dy(i + 2) + dx(i + 3)*dy(i +
        3)
    dtemp = dtemp + dtloc
    60 DDOT = dtemp
RETURN
END
```

Figure 4. Library Subroutine DDOT of SPECchem.

```
DO 310 J = 1, m, mxrows
    jjmax = min(m,j+mxrows-1)
    ij = j*(j-1)/2
DO 300 jj=j,jjmax
DO 200 i = 1,jj
```

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

Volume 5 Issue 2 July 2015

International Manuscript ID : ISSN22308849-V5I2M89-072015

```
ij = ij+1
h(ij)=hij
```

Figure 5. Including Intrinsics in the Symbolic Language.

Compiler can find that the DO 50 loop is parallel. However, subroutine DDOT is called within a triply-nested loop which is also parallel. The parallelism of the outermost loop can be recognized in the situation of the generic DDOT code (the code on the left) but, the compiler is unable to recognize this fact with the transformed code.

Several problems are that hand transformations in endowment codes may have been designed for previous generations of high performance computer systems. For today's machines the transformation may no longer be beneficial or may even degrade performance. SPECseis includes many lower level FFT routines that date back to an IEEE Press book of 1979. These routines are optimized to perform Fourier transforms with minimal memory requirements by writing the output to the supplied input array. Such accumulations introduce memory related dependences, limiting the performance a parallelizing compiler can obtain.

If the compiler is enabled to recognize specific endowment accumulations then the previous accumulations could be undone and the compiler could perform its own. Another approach would be to empower the compiler with the ability to handle all the functions and complexities added by endowment accumulations.

2.3. Symbolic Determination

Parallelizing compiler has its capability to detect data accesses that do or do not access the same memory location. This capability involves the analysis of array subscript expressions. Several compilers in current use on high performance systems can only analyze such expressions if they are affine. Affine subscript expressions contain linear combinations of the iteration variables of enclosing loops. An example from subroutine TFTRI of SPECchem shows the propagated expression used to index an array in loop

```
DO 200:
  x(14+i1+lhc+(jj0*2+(-55)*j1+(11)*jj0+ 25 *j1*
  2)/2+5*j1*jj0) = hij0
```

TFTRI deals with a triangular matrix where the subsequent j(j-1)=2 elements of the work array are accessed in the next iteration. Variables j, jj0, and i1 are loop indices of a triply nested loop. The Polaris parallelizer could propagate and analyze data dependences in the presence of the above polynomial expressions. Therefore, it was able to find the outermost loop of TFTRI, DO 310, to be parallel. However, here found no other compiler with this capability. Codes emphasize the importance of such symbolic analysis techniques.

```
n2 = 1
10 IF (n2 .GE. n .OR. n .EQ. 65536) RETURN
        n2 = 2*n2
        mag = mag+1
        GOTO 10
END
n2 = 2**mag
DO 30 i=1,n2
```

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

Volume 5 Issue 2 July 2015

International Manuscript ID : ISSN22308849-V5I2M89-072015

Figure 6. Recognition of Power-of-2 Loops

In Figure 5, MIN and MAX functions require symbolic analysis to incorporate inequality relations.

Since the size of the data (such as m) is not ensured to be a multiple of mxrows, here need to include such functions as MIN, MAX, MOD, FLOOR, and CEIL in symbolic analysis.

SPECseis poses another challenge to symbolic analysis. Since SPECseis relies heavily on fast Fourier transforms, loops are found that access up to the power of two greater than a dimension of the data. The result of this is that some loops access up to n elements of an array where *n2* is $2^{\lceil \log 2n \rceil}$ of the data size, but Polaris gives up with symbolic analysis when dealing with logarithmic and exponential expressions.

2.4. Array Variations and Type Change

Interprocedural analysis which is above described is a very important technique for dealing with modular programs. Subroutine inline expansions to achieve the same effect are used by the Polaris compiler. A problem that both of these techniques face is that arrays may assume different shapes and have different types in a subroutine and its caller.

2.4.1. Array Variation

The caller routine shapes the array as a D array and the callee shapes it as 1D. No out of bounds indexing occurs by default, this is assumed by FORTRAN compilers. According to this assumption *v(1,i)* and *v(1,j)* in the following example will never overlap as long as i ≠ j. These two portions of the array v are passed as two separate vectors into the DAXPY subroutine, which sees the two parameters as two single dimension arrays. An example of array variations is given in Figure 7.

Multiple problems occur when subroutine DAXPY is in lined into SCHMD. When Polaris inline DAXPY into SCHMD, it linearizes the array index to the D array and accesses v as a one dimensional array. Then, polaris cannot determine that the access to *v(1,i)* which is now *v(i5+(i3-1)*ndim* does not overlap with *v(1,j)* which is now *v(i5+(i3-1+j1)*ndim)*.

```
DO i5 = 1, num0, 1
   v(i5+(i3-1+j1)*ndim) = v(i5+(i3-
   1+j1)*ndim) + v(i5+(i3-1)*ndim)*dum1
ENDDO
```

Another example of variations in Figure 8 shows a situation where portions of a large array declared in the main program of SPECseis are passed into several subroutines.

Caller Routine:

```
SUBROUTINE SCHMD(v,m,n,ldv)
DIMENSION v(ldv,n)
CALL DAXPY(n,dum,v(1,i),1,v(1,j),1)
```

Callee Routine:

```
SUBROUTINE DAXPY(n,da,dx,incx,dy,incy)
DIMENSION dx(*),dy(*)
DO 10 i = 1,n
        dy(iy) = dy(iy) + da*dx(ix)
        ix = ix + incx
        iy = iy + incy
```

Figure 7. Example of Array Variations.

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

Volume 5 Issue 2 July 2015

International Manuscript ID : ISSN22308849-V5I2M89-072015

```
CALL SEICTRI3D
        ( q0(1,1,k), sa(ka), sa(kb), sa(kaa), sa(kbb),
        & sa(kbnx1), sa(kbnxn), sa(kbbnx1),
sa(kbbnxn),
        & sa(kbny1), sa(kbnyn), sa(kbbny1),
sa(kbbnyn),
        & sa(kze), sa(kzf), nx, ny )
```

Figure 8. Array Carving.

In Figure 8, Work array, *sa*, is passed into SEICTRI3D. If inline SEICTRI3D into the caller routine, then the implicit non alias assumption of FORTRAN (the assumption that none of the parameters to a subroutine are aliased) is lost. Only with the non aliasing assumption of FORTRAN 77 know that the accesses to the segments of *sa* do not overlap.

2.4.2. Transformations of Array Type

When the type of an array transforms between the caller and callee subroutine then similar problem arises. Figure 9 gives an example from SPECseis where some arrays are declared real and used as complex within the callee subroutines. This is because it is a large work array where one set of routines use a portion of the work array as a smaller real array and another portion as a smaller complex array.

2.5. Loop Exits and IO Statements

Polaris could not determine that one of the main loops of SPECchem (TWHEIP do#3) is parallel was because of an abort statement. The abort statement is executed only in rare cases but the compiler simply sees a conditional and an exit from the loop. In this case the abort statement was hidden deep in a nest of subroutine calls, loop nests, and conditionals, illustrated in Figure 10. The exits occur only in cases of errors, cases where correct program execution is not applicable then the compiler should ignore program exits when searching for data dependencies. Figure 11 gives another example of conditions that section portions of code.

```
Caller Routine:
        SUBROUTINE MG3D_ZSTEP(...,sa,...)
        This routine propagates wave-field 1 depth
step.
        REAL sa(*)
         Extrapolate in x and y directions
        CALL MG3D_XTRAP(...,sa,sa(ksa))
Callee Routine:
        SUBROUTINE MG3D_XTRAP(...,vel,sa)
        REAL vel(nx,ny)
        COMPLEX sa(*)
```

Figure 9. Array Type Changing

```
SUBROUTINE TWHEIP
        DO iit = 1, npar
        IF (ijkl .EQ. 1) THEN
        CALL GENRAL
            DO 480 kg = 1, ngc
            DO 460 lg = 1, lgmax
           DO 440 n = 1, nij
        IF (nroots .GE. 6) CALL ROOT6P
           DO k = 3,n
               CALL RYSNOD
               CALL ABRT
               CALL abort()
```

International Journal of Enterprise Computing and Business Systems

ISSN (Online) : 2230-8849

Volume 5 Issue 2 July 2015

International Manuscript ID : ISSN22308849-V5I2M89-072015

```
SUBROUTINE RYSNOD
      IF (prod .GE. zero) THEN
            IF (maswrk) WRITE(6,15) m, k
            CALL ABRT
            STOP
      endif
```

Figure 10. Program Exit within a Loop.

```
 IF (kdepth .EQ. 0) THEN
Transpose data if first time through
 CALL JSYNC()
Stored data volume is ( x, f, y ), with y spread across nodes.
Use transpose operations to spread frequencies
     across nodes.
     ( x, f, y ) -> ( x, y, f )
   CALL DTRAN132C( nx, nfp, nyp, ra, sa )
   If up to number of lines, quit
   ELSE IF (kdepth .GE. nz) THEN
        ntro = 0
   IF (node .NE. master) return
   CALL SYSOHDR('MG3D')
   WRITE (ipr,9010) tload, tcomp, tcorr, tcomm, flopsm,
     * atee, ratec, flopsm/(tcomp + tcorr + tcomm)
  return
  endif
```

Figure 11. Rarely Executed Code Sections

These two conditional sections are executed only once per program run, Polaris assumes that could be executed each time this code section is invoked. As a result, Polaris sees a possible call to jsync (synchronize MPI processors), DTRAN132C (Transpose the distributed dimension), a call to SYSOHDR (prints out info to the screen), and a premature return within every invocation of this code section. The value of ntro, which is important for the following seismic routines, is unknown at the end of this code section. (Ntro is the number of traces out of this seismic routine which the next seismic routine in the pipeline will process sees the discussion on the driver routine of SPECseis in Section 2.1.1.

2.6. Large-scale Applications Issues

It is presented that the examples and compiler issues for which the context of large, commercially relevant applications makes a difference. Number of problems faced by compiler that are believed, are equally important in smaller applications. Parallelizing compilers are well capable of analyzing Fortran DO loops for parallelism.

Important general issue is data dependence analysis is needed in the presence of subscripted subscripts and pointers. It is found that subscript arrays are often only written during the initialization of a program or program phase and from then on are constant.

3. CONCLUSIONS

Parallelization is not yet at the level of being fully automatic. Using codes can be seen little success of automatic parallelization. There are clear steps that may be taken to empower automatic parallelizing compilers to produce efficient parallel code. In this paper it has been pointed out several key areas that the compiler community should focus on to enable automatic parallelization to become beneficial to

ISSN (Online) : 2230-8849

Volume 5 Issue 2 July 2015

International Manuscript ID : ISSN22308849-V5I2M89-072015

developers of large scale applications. The first aspect of large applications is the growing amount of modularity. The endowment libraries are another characteristic of the codes. It is important that compilers be able to transform optimizations within endowment codes that no longer apply to modern architectures.

Symbolic analysis becomes increasingly complex with larger application codes. Additionally, expressions contain intrinsic functions, such as logarithms and Modular terms where symbolic analysis and manipulation capabilities need continued improvement. The compiler's ability to deal with this variety of issues is critical for successfully optimizing large scale applications.

Input/output operations are another impediment to successful parallelization. The compiler would have to recognize that certain I/O statements are executed rarely or only in error situations.

The study presented in this paper is only a small step in the direction of understanding the characteristics of large scale applications. Analyzing such applications takes significant effort. Many more, similar studies will be necessary to help the current generation of compilers to become truly useful tools for the user of real world commercial applications.

REFERENCES

1. W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. HoeYinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, B. Pottenger, L. Rauchwerger, and P. Tu. Parallel programming with Polaris IEEE Computer, 29(12):78 82, December 1996.

2. William Blume and Rudolf Eigenmann. An Overview of Symbolic Analysis Techniques Needed for the Effective Parallelization of the Perfect Benchmarks. Proceedings of the International Conference on Parallel Processing, pages II 233 II 238, August, 1994.

3. Rudolf Eigenmann, Insung Park, and Michael J. Voss. Are parallel workstations the right target for parallelizing compilers? In Lecture Notes in Computer Science, No. 1239: Languages and Compilers for Parallel Computing, pages 300 314, March 97.

4. Michael W. Schmidt et. Al. General atomic and molecular electronic structure system. Journal of Computational Chemistry, 14(11):1347 1363, 1993.

5. C. C. Mosher and S. Hassanzadeh. ARCO seismic processing performance evaluation suite, user s guide. Technical report, ARCO, Plano, TX , 1993.