数 値 解 析 38－5
(1991. 10. 19)

IBM RS/6000 のスーパースカラーに適したプログラミング

寒川 光
日本アイ・ビー・エム (株)

IBM RISC System/6000*(RS/6000)は同時並行処理を高度に達成した スーパースカラー・ワークステーションである。この並行性を活かすには、個々の命令実行時間を固定して計算時間を見積るという、既存の方法では殆ど役に立たない。 本論文はまず、RS/6000のアーキテクチャー、ハードウェア の紹介を行い、次にFortranプログラムの計算速度を＇速度評価式＇を通して見積る方法を述べる。この方法に基づくチューニング法での着眼点は既存のスカラー機でのものとかなり異なっているので＇スーパースカラー化＇という言葉を提案する。

NEW APPROACH FOR FORTRAN PROGRAM TUNING ON IBM RS/6000

Hikaru Samukawa
IBM Japan, Ltd.
19-21, Nihonbashi Hakozaki-cho, Chuo-ku, Tokyo 103, Japan

IBM RISC System/6000 (RS/6000,*) is a highly concurrent superscalar workstation. To exploit this high degree of concurrency, conventional method for performance evaluation which assumes constant computational time for each type of instructions is almost meaningless. This paper introduces the architecture and hardware implementation of RS/6000, then describes a new approach to evaluate Fortran program performance running on RS/6000 through 'performance equation'. We propose the term 'superscalarization' for the tuning method based on performance equation, because the focusing point in performance tuning has so largely shifted from that on conventional scalar processors.

(*) RISC System/6000 is a trademark of International Business Machines Corporation.

1. Introduction

A method of performance evaluation of a Fortran program running on a conventional scalar processor might be carried out based on the assumption that computational time is fixed for each type of instructions. However this method is no longer accurate, because performance behavior tends to become complicated as a machine organization becomes complicated. The background of this complexity is from storage hierarchy (cache) and pipeline enhancements. RS/6000 has one of the newest scalar processor, having a capability to compute two floating-point operations in one cycle time. This performance is achieved by the feature called 'superscalar'. An approach of performance evaluation should be modified so that the superscalar performance is exploited.

1.1 Data cache: On machines with storage hierarchy, an execution time of an instruction varies depending on existence of the operand in cache or not. For example, the performance of the following two programs will differ if an available data cache size of the processor is in between 16KB and 160KB.

A) Primed cache : x and y are resident in cache after second call

```
do i=1,1000
 z(i)=ddot(1000,x,1,y,1)
enddo
```

B) Empty cache : x and y should be reloaded into cache for each call

```
do i=1,100
 z(i)=ddot(10000,x,1,y,1)
enddo
```

For performance evaluation and programming tuning, an effect of the data cache situation is to be considered first. By testing simple program like above example, penalty (average delay of an execution time) due to cache-miss can be measured. For RS/6000, it is roughly twenty cycles. It is indispensable to know this number for effective tuning work.

1.2 Processor pipeline: Even in primed cache situation, instruction sequence may affect on the execution time of certain instruction. Even the same sequence, it may vary depending on data. These behaviors can be explained using profound knowledge of how the processor pipeline works. But from a Fortran programmer's point of view, such kind of knowledge will be felt 'too much detail'. So, it is required to establish some simple method to evaluate execution cycles of given Fortran loops through/bypassing instruction sequences.
The following two chapters describe key factors which affect on performance.

- execution overlapping
- depth of dependency

2.RISC System/6000 processor architecture

One of the most notable features of the RS/6000 architecture is the separation of the components of the processor into functional units, i.e., overlapped execution among fixed-point unit, floating-point unit and branch unit can be implemented for this architecture (1).

2.1 Superscalarization: It is introduced in (1) and (2) that the following 2D graphics transform problem can be computed in four cycles per loop iteration on the experimental version of RS/6000 processor (named AMERICA). The given Fortran loop and one of the possible instruction sequences are as follows ;

```
do i=1,n
 xx(i)=b1+a11*x(i)+a12*y(i)
 yy(i)=b2+a21*x(i)+a22*y(i)
enddo

assume fp8=a11, fp7=a21, fp6=a12, fp5=a22, fp4=b1, fp3=b2

lfdu   fp0,r5=x(r5,8)      load x(i), r5=r5+8
fma    fp1=fp4,fp8,fp0     fp1=b1+a11*x(i)
lfdu   fp2,r6=y(r6,8)      load y(i), r6=r6+8
fma    fp0=fp3,fp0,fp7     fp0=b2+a21*x(i)
fma    fp1=fp1,fp6,fp2     fp1=fp1+a12*y(i)
fma    fp0=fp0,fp2,fp5     fp0=fp0+a22*y(i)
stfdu  r7,xx(r7,8)=fp1     store fp1 to xx(i), r7=r7+8
stfdu  r8,yy(r8,8)=fp0     store fp0 to yy(i), r8=r8+8
bc     CL.0,cr1,0x1/lt     branch, decrement count register
```

- lfdu : load floating-point double with update
- stfdu: store floating-point double with update
- fma : floating-point multiply-add
  ('fma fpt=fpb,fpa,fpc' performs 'fpt=fpb+fpa*fpc')
- bc : branch conditional (and count register decrement)
- register renaming : 32 architected registers are mapped onto 40 physical registers.
- floating-point number representation : hexadecimal (AMERICA),
  'fully conforming to the IEEE binary standard' (RS/6000).

All of above instructions are compound-function instructions i.e., multiply and add, load and update, store and update, branch and count. In AMERICA, since load and store are executed by the fixed-point unit, 'fma' is by the floating-point unit and branch is by the branch unit separately, execution times of three units can be overlapped. Multiplication and addition can be overlapped in the floating-point pipeline. This overlapped implementation will only require four cycles for each iteration of the loop.
The concurrency can be simply expressed by introducing a 'performance equation' so that performance of the Fortran program can be estimated without inspecting details inside the processor.

#cycles = max (#load + #store , #'fma')

From the Fortran programs, #load and #store can be counted from the observation of the statements inside of the loop.

- #load : number of unique array elements indexed with the loop index in the right-side of the statements
- #store : number of unique array elements indexed with the loop index in the left-side of the statements

This case, #load is two (x(i),y(i)) and #store is two (xx(i), yy(i)). By substituting 2 for #load, 2 for #store, 4 for #'fma', then 4 cycles is obtained. The former term in the performance equation represents a limitation of the fixed-point unit, the latter does that of the floating-point unit. Function 'max' does the overlapped implementation. Since both terms are equal in this case, execution time of loads and stores is hidden behind that of arithmetic operations. Tuning method of modifying innermost loop to make balancing two terms in performance equation can be called 'superscalarization', which appears in AMERICA in its ideal form.

2.2 Focusing point shifting: The Architecture is contrasted with S/370 architecture using the same program.

Assume B1=b1,B2=b2,A21=a21,A22=a22 and F6=a11,F4=a12

```
TOP   LDR    F2,F6           F2=a11
      MD     F2,0(R9,R8)     F2=F2*x(i)
      AD     F2,B1           F2=F2+b1
      LDR    F0,F4           F0=a12
      MD     F0,0(R9,R7)     F0=F0*y(i)
      ADR    F0,F2           F0=F0+F2
      STD    F0,0(R9,R6)     xx(i)=F0
      LD     F2,A21          F2=a21
      MD     F2,0(R9,R8)     F2=F2*x(i)
      AD     F2,B2           F2=F2+b2
      LD     F0,A22          F0=a22
      MD     F0,0(R9,R7)     F0=F0*y(i)
      ADR    F0,F2           F0=F0+F2
      STD    F0,0(R9,R6)     yy(i)=F0
      BXLE   R9,R10,TOP      branch and increment
```

There are fifteen instructions, and here, assuming that some scalar processors take five cycles for multiplication, three for addition, one for load, two for store and two for branch, more than forty cycles are required for single iteration of the loop. The method of performance evaluation for processors of this type would be accurate enough with assuming fixed number of cycles for each type of instructions (if primed cache situation were provided). Since a ratio of number of cycles of the floating-point arithmetic operations to the rest of the instructions is large, focusing point for tuning work on such processor would be concentrated on arithmetic operations. On the superscalar processor, however, it becomes equally or more important to reduce load and store operations.

3.RISC System/6000 processor pipeline

The actual implementation of production machine (RS/6000) made 'performance equation' more complicated. The most significant change from AMERICA to RS/6000 was that floating-point stores were required to proceed through the floating-point unit (2), i.e., store instruction is executed by both fixed- and floating-point unit. Corresponding to this change, the second term in the performance equation should be revised with adding #store.

#cycles = max(#load + #store , #'fma' + #store)

The 2D graphics example is evaluated to be six cycles from this revised equation. However, actual testing in primed cache situation reveals seven cycles are required (2).

3.1 'fma' dependency: On RS/6000, two types of dependent operations exist. The first type of dependent operations is the one where a source operand depends on the result of a previous floating-point operation (3).

```
                              # additional   execution      depth of
INST.     FRT=FRB+FRA*FRC       cycles       in pipeline    dependency
----------------------------------------------------------------------
 fma      fp1,---,---,---                  D M A W           FRB case
 fma      ---,fp1,---,---       +1           D D M A W          2
----------------------------------------------------------------------
 fma      fp1,---,---,---                  D M A W
 fma      ---,---,fp1,---       +2           D D D M A W
------------------------------------------------------------ FRA case
 fma      fp1,---,---,---                  D M A W              3
 fma      ---,---,---,---                    D M A W
 fma      ---,---,fp1,---       +1             D D M A W
----------------------------------------------------------------------
 fma      fp1,---,---,---                  D M A W           FRC case
 fma      ---,---,---,fp1       +1           D D M A W          2
----------------------------------------------------------------------
```

D ; floating-point decode
M ; floating-point multiplication
A ; floating-point addition
W ; write the result to floating-point register

3.2 Store dependency: The second type of dependent operations is related to stores. When a store enters decoder and the preceding arithmetic operation is still computing the result for the target register content, a subsequent store will stall in the decoder until after the preceding instruction has placed its result in store queue. This is the reason for the 2D graphics example takes one additional cycle over 'six' obtained from the performance equation.

```
                              # additional   execution
INST.     FRT=FRB+FRA*FRC       cycles       in pipeline
--------------------------------------------------------
 fma      fp1,---,---,---                  D M A W
 fma      fp0,---,---,---                    D M A W
 stfdu    fp1                                  D
 stfdu    fp0                   +1               D D
--------------------------------------------------------
```

3.3 Depth of dependency: Thus, in particular instruction sequences, number of execution cycles may increase according to the preceding instructions. The number of 'fma' instructions that influence each other is called 'depth of dependency'. In case of actual tuning work, dependent 'FRA' operands may seldom occur because of the compiler's dependent operation considerations. Consequently, the depth of dependency to be cared is at most 'two'. Depth of two can be often resolved by one-more-way unrolling. The following two-way unrolling is effective in 2D graphics transform problem to remove one additional cycle caused by the store dependency, so that six cycles per one transform is achieved.

```
do i=1,n-1,2
 xx(i  )=b1+a11*x(i  )+a12*y(i  )
 xx(i+1)=b1+a11*x(i+1)+a12*y(i+1)
 yy(i  )=b2+a21*x(i  )+a22*y(i  )
 yy(i+1)=b2+a21*x(i+1)+a22*y(i+1)
enddo
.......................
```

In actual tuning work environment, such tuning margin can be found by measuring kernel program running in primed cache situation.

4.Example of superscalarization

The typical scenario of superscalarization on RS/6000 would be as follows ;

1. adopting block algorithm (making primed cache situation)
2. outer-loop unrolling to reduce loads/stores till minimizing cycles obtained from the performance equation
3. inner-loop unrolling to remove additional cycles caused from dependency

4.1 Example: Matrix-matrix multiplication with size of l, m, n can be blocked so that submatrix of A (ll by mm) fits into an effective cache capacity (4), i.e., two loops are applied stripmining with size of ll and mm.

```
                                     do kk=1,m-1,mm
                                      do ii=1,l-1,ll
do j=1,n                               do j=1,n
 do i=1,l                               do i=ii,min(l,ii+ll-1)
  do k=1,m                               do k=kk,min(m,kk+mm-1)
   c(i,j)=c(i,j)+a(i,k)*b(k,j)            .. = .. +a(i,k) ...
  enddo                                  enddo
 enddo                                  enddo
enddo                                  enddo
                                      enddo
                                     enddo
```

Since the portion of inner three loops of the modified program can work in nearly primed cache situation, further tuning becomes very effective. In this case, #load is two (a(i,k) and b(k,j)), #store is zero. The second term is one. The ratio of the second and the first term (1/2) can be enhanced with applying two-by-two unrolling on j(column) and i(row) loops, so that the number of loads/stores is decreased until balanced with that of the second term.

```
do j=1,n-1,2
  do i=ii,min(l,ii+ll-1)-1,2
```

```
      do k=kk,min(m,kk+mm-1)
       c(i  ,j  )=c(i  ,j  )+a(i,  k)*b(k,j)
       c(i+1,j  )=c(i+1,j  )+a(i+1,k)*b(k,j)
       c(i  ,j+1)=c(i,j+1  )+a(i,  k)*b(k,j+1)
       c(i+1,j+1)=c(i+1,j+1)+a(i+1,k)*b(k,j+1)
      enddo
    enddo
  .............
  enddo
```

In the kernel loop (k-loop), #load is four (a(i,k),a(i+1,k),b(k,j), b(k,j+1)) and #'fma's is four as well. They are balanced and free from dependent operation effects. As a result, eight floating-point operations are carried out in four cycles. Actual testing on RS/6000-550 (41MHz), roughly 74 mflops is measured. Note that final version of tuned program is given finer tuning such as temporary scalar (compiler persuasion) and data compaction (5) applied on submatrix of A (for contiguous access). By using this multiplication routine at the hottest kernel of the recursive block Crout formulation (4), LINPACK problem (TPP) of solving linear equations sized 1000 can be solved with 64 mflops. More sophisticated program (ESSL:Engineering and Scientific Subroutine Library) has a capability to solve this problem with 73 mflops.

4.2 Observations and considerations

Sensitivity: On conventional processors, performance gain would be negligibly small, because an instruction set consisting of two operand instructions inhibits loads/stores reduction. This explicates that performance of a Fortran program is more sensitive to the program modification on superscalar processors than on conventional processors.

Semantic gap: On conventional scalar processors, the number of storage/cache references is difficult to be estimated from the Fortran program. This kind of problem is often called the semantic gap, a measure of the difference between the concepts in high-level languages and the concepts in the computer architecture. RS/6000, on the other hand, can estimate that number for the simple programs exemplified in this paper. This difference resulted from the efforts to minimize the performance impact of the data transfer from storage/cache to registers, carefully made by the architects of RS/6000 in search for the ultimate concurrency. To be more specific,

- 'fma' having four register operands and enough number of registers reduce storage/cache accesses,
- The sophisticated way of operand address update simplifies scaler pipeline design.

These items not only are the key factors for performance optimization, but also play an important role to shrink the semantic gap, which is helpful for Fortran programmers to be able to evaluate their execution performance in terms of possible program modifications such as by unrolling. And the importance of code movement by the compiler cannot be looked over as well.
It is added that this optimized architecture is named POWER (Performance Optimized With Enhanced RISC) architecture.

5.Comments

5.1 Register renaming: The floating-point register cannot be overwritten until all prior floating-point instructions which refer the old value of the register have accessed that value. The capability of the register renaming resolves the interlock caused by this floating-point register conflict, which often appears in tight loop. And, even in high degree of overlapped implementation, a precise interruption is still held by the register renaming.

5.2 Data dependent performance: Computational time for some floating-point numbers that require reserved exponent expression such as denormalized numbers is longer than normalized numbers.

5.3 Relation to the Fortran compiler capability: According to the XL Fortran rule for constructing arithmetic expressions, since order of multiple additions in single executable statement is from left to right (6). In case of 2D graphics transform example, if the statement is written in the form 'xx(i)=a11*x(i)+a12*y(i)+b1', one of the 'fma's splits into 'multiply' and 'add' instructions. XL Fortran preprocessor provides a capability called associative transformation which automatically changes expressions to the form 'xx(i)=b1+a11*x(i)+a12*y(i)'.
Other preprocessor functions such as 'unrolling' and 'stripmining' are provided to help automatic program modifications exemplified in the previous section.

6.Conclusions: A method of performance evaluation is substantially architecture/hardware dependent. An effective approach to tune a Fortran program running on the machines with storage hierarchy and superscalar processor is shifted from that on conventional scalar processors. For the storage hierarchy, knowing the magnitude of cache-miss penalty and adopting blocking algorithms are effective. For the superscalar, evaluating performance through the performance equation in order to take overlapped execution into account, and reducing loads/stores by unrolling are effective. Then measure in actual run in the primed cache situation to find additional cycles. If found, apply unrolling again to remove these additional cycles. Since this procedure is specific to superscalar processors, it can be called 'superscalarization'.

Acknowledgments: We are grateful to Brett Olsson, Steve White and Richard Fry (IBM/Austin) for providing us a detailed information of RS/6000 processor.

References

1. R.R.Oehler and R.D.Groves,"IBM RISC System/6000 Processor Architecture," IBM J.Res.Develop.34,23-36(1990)
2. G.F.Grohoski,"Machine organization of IBM RISC System/6000 Processor," IBM J.Res.Develop.34,37-58(1990)
3. B.Olsson,R.Montoye,P.Markstein and M.NguyenPhu,"RISC System/6000 Floating-Point Unit," IBM RISC System/6000 Technology, Order Number SA23-2619(1990), available through IBM branch offices.
4. H.Samukawa,"Programming style on the IBM 3090 Vector Facility considering both performance and flexibility," IBM Systems Journal 27, No.4,453-474(1988)
5. B.Liu and N.Strother,"Programming in VS Fortran on the IBM 3090 for Maximum Vector Performance," IEEE COMPUTER, JUNE, 65-76(1988)
6. "AIX XL FORTRAN Compiler/6000 Version 2 Language Reference" and " User's Guide," Order Number SC09-1353 and -1354(1991), available through IBM branch offices.