경영과학의 응용
제2권,1985년 8월

대형최적화모델의 소형 COMPUTER 적용을 위한 효율적인 IN-CORE STORAGE ALLOCATION

금성사 시스템사업부
본 부 장(工博) 안 규 호

1. 서

최근 Mini 및 Micro Computer 산업의 급속한 발전은 상대적으로 저렴한 가격의 Computer System 보급이 가능하게 되어 Mini 및 Micro computer의 확산은 증가하는 추세에 있다. 또한 일반적으로 Mathematical Programming 기법으로 접근하는 real-world 최적화 모델은 결정 변수나 제약 조건의 수로 볼때 대형화하고 있다.

많은 최적화 문제가 Analytical한 접근보다는 Iterative한 접근에 의해 최적해를 구하고 있으며 따라서 특히 Large Scale Optimization Problem의 경우 많은 양의 Data를 효율적으로 Update하고 처리하기 위한 Algorithm 및 Data structure는 중요한 연구분야라 하겠으며 각 Model의 특성을 이용한 Specialized Algorithm 대해 많은 연구가 행해지고 있다.

일반적으로 disk access를 통한 I/O time은 CPU time에 비해 막대한 시간이 소요되므로 계산에 관련되는 모든 Data(특히 매 iteration 마다 update되는 부분)를 in-core storage에 가지고 처리하는 CPU-Bound Job이 효율적인 것은 두말할 필요도 없다.

그러나 Computer System의 resource 로서의 in-core memory는 제한이 있기 마련이다. Physical Memory의 제한은 별도로 하더라도 특정한 Computer에서 addressing 가능한 user memory space는 각 computer의 architecture에 따라 천차만별이다. (표 1 참조)

아래 표에서와 같이 Computer Architecture 측면에서 보아 Virtual Memory 개념이 없는 기종이나 Fortran Compiler상 제약이 있는 기종에서는 이 Memory의 사용 제한을 벗어나 좀 더 많은 memory space를 확보할 수 있는가, 그리고 확보된 Memory를 어떻게 효율적으로 사용하는가가 선결 사항으로 대두된다.

기 종		Word length	Physical Memory (최대)	Addressing Capacity	비 고
Super	VAX 11/780	32 bit	64 MB	4 GB	Virtual Memory
Mini	MV 10000	32 bit	16 MB	2GB/USER	〃
Mini	DPS6 Series	16 bit	16 MB	128 KB	
	PDP 11	16 bit	4 MB	256 KB	
Micro	IBM PC/XT	16 bit	640 KB		FORTRAN Compiler의 제약

〈표 1〉

사용 가능한 User memory의 제한을 극복못하는 경우는 Data matrix의 대부분이 Disk에 있게 되므로 Large Scale Optimization Problem은 필연적으로 CPU Processing보다는 매 iteration마다 disk accress를 통한 I/O Processing time이 더 많이 걸리는 I/O Bound Job으로 그 속성이 전도되어 Real Problem의 실행 시간이 수시간 또는 수일로 지연되어 실용 가능성의 한계를 넘게 된다.

따라서 실제 보급이 확산되고 있는 Mini 및 Micro Computer에서 Large Scale Optimization Problem의 해를 비교적 빠른 시간내에 구할 수 있도록 사용 가능 In-core Memory Space의 확장 및 그 효율적 활용 방안을 논의하고자 한다.

2. Memory의 효율적인 사용방안

2·1 Main Array의 정의

User가 사용가능한 Memory Space의 범위내에서 효율적으로 Storage를 사용하기 위해서는 기본적으로 모든 work space array의 size가 문제의 크기에 따라 변할 수 있도록 정의되어야 한다. 또한 임의의 문제의 크기에 따라 변하는 array size는 Program의 recompilation없이 처리되어 특정 기계의 사용 가능한 memory space를 최대한 이용하도록 storage를 분할하여야 한다.

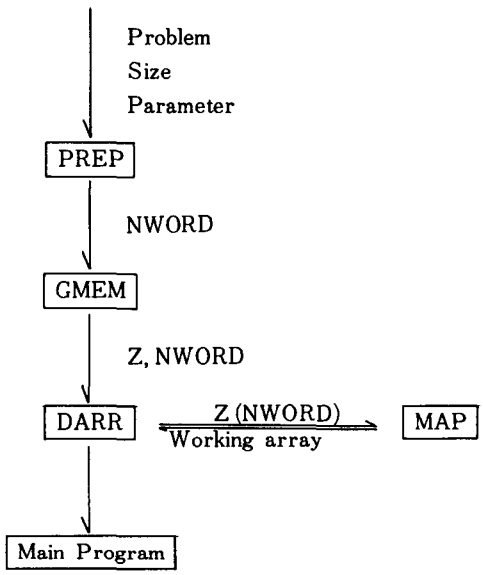
Program 내의 work space array의 크기는 문제 크기를 결정해 주는 여러개의 paramter들에 의해 정의될 수 있다. 따라서 모든 work space array는 하나의 1차원 Main array Z(*)를 정의하여 필요한 크기만큼 분할하여 사용한다. 즉 Array Z(*)의 type은 특정기계의 가장 큰 word size로 정의하며 다른 array들은 기계에서 사용가능한 word size를 정의하는 상수를 이용해 Z(*) 내에 Compact하게 분할하여 사용한다.

위의 방법을 택함으로써 임의의 parameter들 값에 대해 어떤 work space array도 필요한 크기이상 정의되는 것을 피할 수 있고 문제의 크기(parameter 값)에 따른 array Z(*)의 적정 size를 계산하여 정의하게 된다. 이러한 Mainarray Z(*)는 Assembly routine를 통하여 recompilation 없이 size가 정의되며, 따라서 기종에 따른 Fortran array size의 제약, 제한된 Memory Space의 확장등이 가능하게 된다. 그리고 Multi-Task/Multi-user job이 수행될때 각 job은 적정량의 memory만을 사용하게 되어 in-core memory를 효율적으로 사용할 수 있게 된다.

2.2 제한된 Memory Space의 확장

One User가 Addressing 가능한 memory space가 제한되는 것은 각 Computer의 architecture에 따라 addressing에 참여하는 bit 수가 얼마나 되는냐에 따른 것으로써 SourceProgram의 Compiler & Link시에 System이 generate 해주는 data와 code area가 maximum addressing space를 넘을수 없기 때문이다.

따라서 이 제한을 벗어나기 위해 다음과 같은 절차를 거친다. (그림 2 참조)



〈그림 2 : 유통도〉

(1) PREP : NWORD 계산
- Problem size에 관련된 Parameter 의 입력
- Main Array Z(*)에 필요한 8 bytes element의 수효 "NWORD" 계산
- Assembly routine "GMEM"을 call하여 NWORD를 pass

(2) GMEM : Assembly Program으로 Memory 확보
- PREP로 부터 NWORD 입력
- 필요한 memory space를 확보
- Main array Z(*)의 이름과 크기(Z, NWORD)를 DARR에 pass

(3) DARR : Main array 및 work space array 정의
- Main array Z(NWORD) 정의
- Subroutine "MAP"을 call 하여 관련된 work space array를 위해 Z(*)를 분할
- 정의된 array들을 Main Program에 pass

```
PROGRAM PREP
INTEGER*4 NWORD
COMMON / SIZPAR / L1, M1, N1,

READ*, L1, M1, N1
NWORD = M2 *N2 +10*M1 +L1 +30

CALL GMEM(NWORD)
STOP
END
```

〈표 3 PREP의 예〉

```
SUBROUTINE DARR (Z, NWORD)

INTEGER*4 NWORD
DOUBLE PRECISION Z(NWORD)

COMMON / SIZPAR / L1, M1, N1
COMMON / LMAP 1 / LA, LHA, LKA
COMMON / LAMP 2 / ·····
    .
    .
CALL MAP (L1, M1, N1)
CALL MAIN(Z, NWORD, Z(LA), Z(LHA), M1·····)
    .
    .
    .
RETURN
END

SUBROUTINE MAP (L1, M1, N1)

COMMON / LMAP 1 / LA, LHA, LKA
COMMON / LMAP 2 / ·····
    .
    .
NWORDI= 2
NWORDH= 4
NWORDR= 2

LA= 1
LHA=LA+N1 *M1 + 1
LKA=LHA+ (M1 / NWORDH) + 1
    .
    .
    .
RETURN
END

SUBROUTINE MAIN(Z, NWORD, A, HA, M1,·····)

DOUBLE PRECISION Z(NWORD), A(M1)
INTEGER*2          HA(M1)
    .
    .
    .
RETURN
END
```

〈표 5 DARR, MAP의 예〉

```
                TITLE       GMEM
                LIBM        EXEC-LIB
                XDEF        PREP
                XLOC        DARR
                EQU         $
*
                SAVE        SAVER, Z 'FFFF'
                LDR         $R5, +$B7
*
                LDB         $B3, $B7
                LDI         $B3
                SDI         NC1
                MLV         $R7, =4 → NWORD*4 Words
                LDV         $R2, =0
                MCL                          →Get Memory
                DC          Z '0402'           Macro call
                BNEZ        $R1, RPTER
*
                STB         $B4, Z1
*
                LAB         $B7, PARBLK
                LNJ         $B5, DARR → Jump to DARR
*
                LDB         $B4, Z1
                MCL
                DC          Z'0404'
                BNEZ        $R1, RPTER
*
END1            RSTR        SAVER, Z '7FFF'
                JMP         $B5
RPTER           $RPTER
                B           >END1
*
PARBLK          DC          7
Z1              RESY        2,0
N1              DC          <NC1
NC1             RESV        2,0
*
SAVER           RESV        23
                CTRL        LINK      DARR
                END
```

〈표 4 GMEM의 예〉

이상과 같은 절차를 실제 간단한 예로써 제시하면 표 3 ~표 5와 같다. 표 3에서 문제의 크기에 관련된 parameter L1, M1, N1이 입력되면 전체 work array에 필요한 8 bytes element 의 수 NWORD를 계산하여 GMEM으로 pass한다. 표 4에서 ARRAY Z(NWORD)의 space를 확보하고 표 5의 Subroutin MAP에서는 WORK ARRAY A, HA, KA등의 Z(NWORD) 분할사용시 각각의 Starting point LA, LHA, LKA 등이 계산된다.

예를 들면 HA는 size M1인 half-interger array인 경우이다. GMEM에서 memory를 확보하는 것은 PREP에서 pass된 NWORD 수만큼 Double precision으로 확보하므로 그 최대 SPACE는 필요시 8 bytes $*(2^{31}-1)=16$GB 까지 가능하다.

즉 Physical Memory가 허용하는한 memory space를 사용할 수 있는셈이 된다. Subroutine MAP에서 일부 work space array는 입력된 parameter에 따라 fix 할수 없는 경우도 있다. 예를들어 LP에서 Basis matrix의 nonzero 의 수가 지속적으로 증가하는 경우 일차적으로 re-inverion으로 해결하지만 그 빈도가 높은 경우는 execution 상에서 Z(NWORD)를 확장시킬 필요가 있다.

이 경우는 적정량의 NONZERO 원소의 수를 정의한 후 실제 Program 상에서 Z(WORD)가 모자라는 경우 그 시점에서 PREP로 return시켜 NWORD를 새로 정의함으로써 해결된다.

3. 결 론

이상과 같이 virtual memory 개념이 없는 mini/micro computer에서 Fortran compiler 의 array size 제약이나 one-user가 사용 가능한 memory space의 제약을 벗어나고 실제 문제의 크기에 적합한 크기의 incore memory를 source program의 recompilation 없이 효율적으로 사용할 수 있도록 Single array Z(*)의 도입과 Assembly routine과의 결합에 대해 살펴보았다.

이러한 기법은 실제 중소형 computer에서 OR package를 개발하는 과정에서 발생하기 쉬운 in-core memory상의 문제점을 해결해 나가는 간단하고 효율적인 방법이라고 사료된다.