Lecture 17 - More on Parallel/Processor Domain Decomposition Techniques

- **In our last lecture, we discussed general ways to perform parallel/processor domain decomposition for different types of problems**
- **Again, in engineering simulations, we usually generate multiple blocks of structured- or unstructured-grids to solve a (set of) ODE's or PDE's that govern the problem physics**
- **Assuming we have a computer code that will solve these governing equations on a parallel computer, the next step is to map the blocks of grids onto P processors of the parallel computer**

Mapping Techniques

- **There are numerous techniques to map blocks (nodes, or cells) onto processors**
  - First-come-first-serve (i.e. in order of stack) or even distribution (naïve)
    - Trouble is that all blocks may not be the same! Some blocks may have more nodes (elements) than others. (naïve)
  - Geometrically constrained distribution of blocks
    - Distribute blocks such that we attempt to evenly distribute the number of nodes (elements) to the processors. (much better!)
  - Communication and geometrically constrained distribution of blocks
    - Distribute blocks such that we not only attempt an even distribution of nodes (elements) but also attempt to evenly distribute the inter-processor communication cost. (even better!)
- **Let's assume we are mapping blocks as we will do in the class project**

First-Come-First-Serve or Even Distribution

- **If we distribute N blocks over P processors, then the average number of blocks per processor will be N/P**
  - There are different possible distributions (some better than others)
    - First-come-first-serve: Take the first block and give it to the first processor, the second block to the second processor, and so on….. (more random, naïve)
      - This approach may be OK in some situations where a random distribution would more evenly load-balance the processors (but not for our project)
    - Even distribution: Take the first N/P blocks and assign them to the first processor, the second N/P blocks to the second processor, and so on. Take the remainder and sequentially distribute them over some part of P processors. (better!)
      - This approach would be much better in our project since our block list is contiguous and it would result in less inter-processor communication.
    - A measurement of the "load-balance" of a processor will be (M*P)/N where M is the actual number of blocks on a given processor, P is the total number of processors, and N is the number of blocks

First-Come-First-Serve or Even Distribution

- **This is equivalent to creating a task-dependency graph with no weights associated with the tasks**
  - All blocks are assumed to have the same number of nodes (cells) and therefore, have the same operation count and weight
  - Grouping of blocks onto processors would be performed in a way to evenly distribute the number of blocks amongst the processors
  - No accounting of communication cost is made

Geometrically Constrained

- **Here, we could distribute the blocks such that an attempt is made to give each processor approximately the same number of nodes or cells**
- **This can also be done in a number of ways:**
  - Pre-sorting:
    - Determine the “perfect load-balance”,PLB= (total number of nodes (cells)/P)
    - Sort the blocks in terms of number of nodes (cells) from maximum to minimum.
    - Then distribute the sorted blocks sequentially to the processors with the least number of nodes (cells) (i.e. take the next block in the sorted list and assign it to the processor that has the least number of nodes (cells) thus far)
    - A measurement of the “load-balance” of a processor will be M/PLB where M is the actual number of nodes (cells) on a given processor and PLB is the “perfect load-balance”
  - Non-pre-sorted:
    - You could do the same steps without pre-sorting but you are more likely to end up with a worse “load-balanced” system

Geometrically Constrained

- **Geometrically constrained mapping would be equivalent to again creating a task-dependency graph. But now, the weight of each task would depend on the number of nodes (cells).**
  - Grouping of blocks onto processors would be performed in a way to evenly distribute the “weight” amongst the processors
  - Again, no accounting of communication cost is made

Example Code for Geometrically Constrained Mapping

```
C   *****************************************************************
C   NAME=LOADBAL
    SUBROUTINE LOADBAL(NTCELL,NBL,JPROC,NPROC,NGDOM,NBLIST,NBPROC)
C   *****************************************************************
C   *                                                               *
C   *  ASSIGNS BLOCKS TO PROCESSORS FOR OPTIMUM LOAD BALANCE  *
C   *                                                               *
C   *****************************************************************
    IMPLICIT REAL*8 (A-H,O-Z)
    INTEGER   NTCELL, NPROC, NGDOM
    DIMENSION NBL(3,*), JPROC(*), NBPROC(*)
    DIMENSION NBLIST(NPROC,*)
C   NTCELL    =  TOTAL NUMBER OF CELLS
C   NBL       =  ARRAY OF DIMENSIONS PER BLOCK (1,NB) = IDIMENSION
C                (2,NB) = JDIMENSION, (3,NB) = KDIMENSION
C   NPROC     =  NUMBER OF PROCESSORS
C   NGDOM     =  NUMBER OF BLOCKS IN GLOBAL DOMAIN
C
C   LOCAL SPACE
    INTEGER ISIZE(NGDOM), ICLAIM(NGDOM), ISORT(NGDOM)
    INTEGER NCELL(NPROC)
C
C   FIND THE SIZE OF EACH BLOCK AND MARK BLOCKS UNCLAIMED
    DO NB=1,NGDOM
C   CALCULATE THE NUMBER OF CELLS FOR THIS BLOCK (2 GHOST CELL LAYERS)
     ISIZE(NB)  = (NBL(1,NB)+3)*(NBL(2,NB)+3)*(NBL(3,NB)+3)
     ICLAIM(NB) = 0
    END DO
    DO N=1,NPROC
     NCELL(N) = 0
    END DO
C
C   FIND SIZE OF PERFECT LOAD BALANCE
    FNTEVEN = FLOAT(NTCELL)/FLOAT(NPROC)
C
C   SORT BLOCKS IN ORDER OF DECREASING SIZE
    DO NB=1,NGDOM
     IMAXSIZE = 0
     DO I=1,NGDOM
      IF (ISIZE(I).GT.IMAXSIZE .AND. ICLAIM(I).EQ.0) THEN
       IMAXSIZE = ISIZE(I)
       IMAX     = I
      END IF
     END DO
     ICLAIM(IMAX) = 1
     ISORT(NB)    = IMAX
    END DO
C
C   ASSIGN BLOCKS TO PROCESSORS IN ORDER TO KEEP MAXIMUM TO A MINIMUM
    DO NB=1,NGDOM
     IMINSIZE = 10000000
C   FIND THE PROCESSOR WITH THE MINIMUM NUMBER OF CELLS
     DO I=1,NPROC
      IF(NCELL(I).LT.IMINSIZE) THEN
       IMIN = I
       IMINSIZE = NCELL(I)
      END IF
     END DO
C   ASSIGN THE CELLS OF THIS NB BLOCK TO THAT PROCESSOR
     NCELL(IMIN)      = NCELL(IMIN) +ISIZE(ISORT(NB))
     JPROC(ISORT(NB)) = IMIN
    END DO
```


Example Code for Geometrically Constrained Mapping (cont)

```
C
C   OUTPUT LOAD BALANCE INFORMATION
    WRITE(*,*)
    WRITE(*,*)'LOAD BALANCE INFORMATION'
    WRITE(*,*)
C
    FLBMAX = -1.E10
    FLBMIN = +1.E10
    DO NP=1,NPROC
     FLB = FLOAT(NCELL(NP))/FNTEVEN
     IF (FLB.GT.FLBMAX) FLBMAX = FLB
     IF (FLB.LT.FLBMIN) FLBMIN = FLB
     WRITE(*,*)'PROCESSOR',NP,'   LOAD PERCENTAGE=',FLB
    END DO
    WRITE(*,*)
    WRITE(*,*) 'MAX/MIN LOAD = ',FLBMAX,FLBMIN
    WRITE(*,*) 'LOAD BALANCE RATIO:',FLBMAX/FLBMIN
    WRITE(*,*)
    IF (FLBMAX/FLBMIN .GT. 1.2) THEN
     WRITE(*,*)
     WRITE(*,*) 'WARNING: WARNING: WARNING: WARNING: WARNING:'
     WRITE(*,*) 'The load is not balanced well with the'
     WRITE(*,*) 'current decomposition and # of processors.'
     WRITE(*,*) 'Consider re-decomposition or reduce number'
     WRITE(*,*) 'of processors.'
     WRITE(*,*)
    END IF
C
C   WRITE OUT BLOCKS FOR EACH PROCESSOR
    DO NP=1,NPROC
     NN = 0
     DO I=1,NGDOM
      IF (JPROC(I).EQ.NP) THEN
       NN = NN +1
       NBLIST(NP,NN) = I
      END IF
     END DO
     NBPROC(NP) = NN
     WRITE(*,*)'PROCESSOR',NP,'BLOCKS',(NBLIST(NP,I),I=1,NN)
    END DO
    WRITE(*,*)
C
    DO NP=1,NPROC
     WRITE(*,*)'PROCESSOR',NP,'NCELLS',NCELL(NP)
    ENDDO
C
    RETURN
    END
```

Communication and Geometrically Constrained

- **Here, we could distribute the blocks such that an attempt is made to give each processor approximately the same communication cost as well as the number of nodes or cells**
- **This can also be done in a number of ways (depending on how you wish to account for communication costs):**
  - Pre-sorting:
    - Determine the “perfect load-balance”,PLB= (total “weight”/P)
    - Sort the blocks in terms of a “weight” from maximum to minimum where the “weight” is a blended formula of the number of nodes (cells) and communication cost. Communication cost could be approximated as the number of nodes along the block faces.
    - Then distribute the sorted blocks sequentially to the processors with the least “weight” (i.e. take the next block in the sorted list and assign it to the processor that has the least accumulated “weight” thus far)
    - A measurement of the “load-balance” of a processor will be M/PLB where M is the actual accumulated “weight” on a given processor and PLB is the “perfect load-balance”
- **You can see that this is a generalization of the geometrically constrained mapping**

Communication and Geometrically Constrained

- **Communication and geometrically constrained mapping would be equivalent to creating both task-dependency and task-interaction graphs. Now, the weight of each task would depend on the number of nodes (cells) and the weight of each path between interactions would depend on the communication cost.**
  - Grouping of blocks onto processors would be performed in a way to evenly distribute the total “weight” based upon the number of nodes (cells) and estimate of communication cost across edges amongst the processors

Example Code for Communication and Geometrically Constrained Mapping

```
C    *****************************************************************
C    NAME=LOADBAL_JCV
     SUBROUTINE LOADBAL_JCV(NTCELL,NBL,ITYPE,NEIGHB,INTRF,JPROC,
   &                       NPROC,NGDOM,NBLIST,NBPROC)
C    *****************************************************************
C    *  ASSIGNS BLOCKS TO DIFFERENT PROCESSORS IN ORDER TO          *
C    *   OPTIMIZE LOAD BALANCE, INCLUDING COMMUNICATION INFLUENCES   *
C    *****************************************************************
C    MODIFIED BY:  JC VASSBERG  16 APR 1997  TO INCLUDE COMM INFLUENCE.
     IMPLICIT REAL*8 (A-H,O-Z)
C   NTCELL    =  TOTAL NUMBER OF CELLS
C   NBL       =  ARRAY OF DIMENSIONS PER BLOCK (1,NB) = IDIMENSION
C                (2,NB) = JDIMENSION, (3,NB) = KDIMENSION
C   NPROC     =  NUMBER OF PROCESSORS
C   NGDOM     =  NUMBER OF BLOCKS IN GLOBAL DOMAIN
C
     INTEGER   NTCELL,NPROC,NGDOM
     DIMENSION NBL(3,*),JPROC(*),NBPROC(*)
     DIMENSION ITYPE(6,*),NEIGHB(6,*),INTRF(6,*)
     DIMENSION NBLIST(NPROC,*)
C
C    LOCAL SPACE
     INTEGER ITABLE(NPROC,NPROC),ICLAIM(NGDOM),ISORT(NGDOM)
     REAL*8  SIZE(NGDOM),FNCELL(NPROC)
C
C    *****************************************************************

C    FIND THE SIZE OF EACH BLOCK AND MARK BLOCKS UNCLAIMED
C    ADD TO THIS SIZE A VIRTUAL SIZE RELATED TO COMMUNICATIONS
C       FNOPS DEPENDS ON CODE, FLOPS, BANDWIDTH, AND LATENCY DEPEND
C       ON COMPUTER SYSTEM
C    FNOPS     - NUMBER OF FLOATING POINT OPERATIONS PER POINT REQUIRED
C              TO COMPLETE A SINGLE ITERATION
C    FLOPS     - FLOPS RATING FOR THE PROCESSORS (REALISTIC FLOATING
C              POINT OPERATIONS PER SECOND)
C    BANDWIDTH - BYTES PER SECOND FOR MPI COMMUNICATION
C    LATENCY   - MPI TIME FOR A MESSAGE OF ZERO LENGTH
C    BANDWIDTH AND LATENCY CAN BE FOUND USING BOUNCE
C
C    SP2:      FLOPS     = 50.0E+6 (FLOATING POINT OPS. / SEC)
C              BANDWIDTH = 35.0E+6 (BYTES / SEC)
C              FLATENCY  = 43.0E-6 (SEC)
C
C    ETHERNET:  FLOPS     = 50.0E+6 (FLOATING POINT OPS. / SEC)
C              BANDWIDTH = 0.1E+6  (BYTES / SEC)
C              FLATENCY  = 800.E-6 (SEC)
C
     FNOPS     = 2000.0
     FLOPS     = 50.0E+6
     BANDWIDTH = 35.E+6
     FLATENCY  = 45.E-6
     CBNDWD    = (120. *FLOPS)/(BANDWIDTH *FNOPS)
     CLATNC    = (5.0 *FLATENCY *FLOPS)/FNOPS
C
```


Example Code for Communication and Geometrically Constrained Mapping

```
C
 WRITE(*,*)
     WRITE(*,*) 'THE CURRENT COMMUNICATION PARAMETERS ARE FOR SP2.'
     WRITE(*,*)
     WRITE(*,*)
     WRITE(*,*) 'LOADBAL WEIGHTS'
     WRITE(*,*)
     WRITE(*,*) 'WEIGHT ON NUMBER OF CELLS =',1.0
     WRITE(*,*) 'WEIGHT ON BANDWIDTH      =',CBNDWD
     WRITE(*,*) 'WEIGHT ON LATENCY        =',CLATNC
     WRITE(*,*)
C
C    ASSIGN A SIZE TO EACH DOMAIN BASED ON ITS FLOP & BANDWIDTH
C      REQUIREMENTS.
     DO NB=1,NGDOM
C     COMPUTE DIMENSIONS OF EACH BLOCK (2 GHOST CELL LAYERS)
      IDIM     = NBL(1,NB) +3
      JDIM     = NBL(2,NB) +3
      KDIM     = NBL(3,NB) +3
C    SIZE IS NOW A FUNCTION OF NUMBER OF CELLS+BANDWIDTH WEIGHT
C     ASSOCIATED WITH FACE COMMUNICATION  (NOTE THAT YOU CAN MAKE
C     SIZE A FUNCTION  OF DIFFERENT PARAMETERS
      SIZE(NB)   = DBLE(IDIM*JDIM*KDIM)
    &           +4.*CBNDWD*
    &           DBLE(IDIM*JDIM +IDIM*KDIM +JDIM*KDIM)
      ICLAIM(NB) = 0
      JPROC (NB) = 0
     END DO

C
C    INITIALIZE ADDITIONAL ARRAYS
     DO N=1,NPROC
      FNCELL(N)    = 0.0
      DO I=1,NPROC
       ITABLE(I,N) = 0
      END DO
      ITABLE(N,N)   = 1
     END DO
C
C    FIND SIZE OF PERFECT LOAD BALANCE W/O COMMUNICATION OVERHEAD
     FNTEVEN = DBLE(NTCELL)/DBLE(NPROC)
C
C    SORT BLOCKS IN ORDER OF DECREASING SIZE
     DO NB=1,NGDOM
      AMAXSIZE = 0.0
      DO I=1,NGDOM
       IF (SIZE(I).GT.AMAXSIZE .AND. ICLAIM(I).EQ.0) THEN
        AMAXSIZE = SIZE(I)
        IMAX     = I
       END IF
      END DO
      ICLAIM(IMAX) = 1
      ISORT(NB)    = IMAX
     END DO
```

Example Code for Communication and Geometrically Constrained Mapping

```
C
C    ASSIGN BLOCKS TO PROCESSORS IN ORDER TO KEEP THE RESULTING
C     MAX-LOADED PROCESSOR TO A MINIMUM
      DO NB=1,NGDOM
       NBLK     = ISORT(NB)
       IDIM     = NBL(1,NBLK) +3
       JDIM     = NBL(2,NBLK) +3
       KDIM     = NBL(3,NBLK) +3
       IJKD     = IDIM*JDIM*KDIM
       IMIN     = 1
       AMINSIZE = 1.0E10
       DO I=1,NPROC
        ATMPSIZE = FNCELL(I) +SIZE(NBLK)
        DO LSIDE = 1,6
         LDIR = (LSIDE +1)/2
         NABR = NEIGHB(LSIDE,NBLK) +INTRF(LSIDE,NBLK)
         IF (NABR .EQ. 0) THEN
          ATMPSIZE = ATMPSIZE
     .              -2.*CBNDWD*DBLE(IJKD/(NBL(LDIR,NBLK) +3))
         ELSE
          JPNABR = JPROC(NABR)
          IF (JPNABR .EQ. I) THEN
           ATMPSIZE = ATMPSIZE
     .       -4.*CBNDWD*DBLE(IJKD/(NBL(LDIR,NBLK) +3))
          ELSE
           IF (ITABLE(I,JPNABR) .EQ. 0) THEN
             ATMPSIZE = ATMPSIZE + CLATNC
           ENDIF
          ENDIF
         ENDIF
        END DO
C
        IF (ATMPSIZE .LT. AMINSIZE) THEN
         IMIN     = I
         AMINSIZE = ATMPSIZE
        END IF
       END DO
C
       FNCELL(IMIN) = AMINSIZE
       JPROC (NBLK) = IMIN
C
C     FLAG THAT WE ARE COMMUNICATING BETWEEN THE JPNBLK & JPNABR
C        PROCESSORS.
       JPNBLK   = JPROC(NBLK)
       DO LSIDE = 1,6
        NABR   = NEIGHB(LSIDE,NBLK)
        IF (NABR .GT. 0) THEN
         JPNABR   = JPROC(NABR)
         IF (JPNABR .GT. 0) THEN
          IF (ITABLE(JPNBLK,JPNABR) .EQ. 0) THEN
            FNCELL(JPNABR) = FNCELL(JPNABR) +CLATNC
          ENDIF
          ITABLE(JPNBLK,JPNABR) = 1
          ITABLE(JPNABR,JPNBLK) = 1
         ENDIF
        ENDIF
       END DO
      END DO
```


Example Code for Communication and Geometrically Constrained Mapping

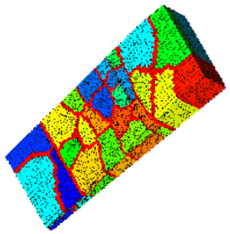
```
C
C    OUTPUT LOAD BALANCE INFORMATION
      WRITE(*,*)
      WRITE(*,*)'LOAD BALANCE INFORMATION'
      WRITE(*,*)
C
      FLBMAX = -1.E10
      FLBMIN = +1.E10
      DO NP=1,NPROC
       FLB = FNCELL(NP)/FNTEVEN
       IF (FLB.GT.FLBMAX) FLBMAX = FLB
       IF (FLB.LT.FLBMIN) FLBMIN = FLB
       WRITE(*,*)'PROCESSOR',NP,'   LOAD PERCENTAGE=',FLB
      END DO
      WRITE(*,*)
      IF (FLBMAX/FLBMIN .GT. 1.2) THEN
       WRITE(*,*)
       WRITE(*,*) 'WARNING: WARNING: WARNING: WARNING: WARNING:'
       WRITE(*,*) 'The load is not balanced well with the'
       WRITE(*,*) 'current decomposition and # of processors.'
       WRITE(*,*) 'Consider re-decomposition or reduce number'
       WRITE(*,*) 'of processors.'
       WRITE(*,*)
      END IF
C
C
C    WRITE OUT BLOCKS FOR EACH PROCESSOR
      DO NP=1,NPROC
       NN = 0
       NC = 0
       DO I=1,NGDOM
        IF (JPROC(I).EQ.NP) THEN
         NN = NN +1
         NBLIST(NP,NN) = I
         IDIM = NBL(1,I) +3
         JDIM = NBL(2,I) +3
         KDIM = NBL(3,I) +3
         IJKD = IDIM*JDIM*KDIM
         NC   = NC + IJKD
        END IF
       END DO
       FNCELL(NP)  = NC
       NBPROC(NP) = NN
       WRITE(*,*)'PROCESSOR',NP,'BLOCKS',(NBLIST(NP,I),I=1,NN)
      END DO
c
      WRITE(*,*)'FNTEVEN',FNTEVEN
      DO NP=1,NPROC
       NC = FNCELL(NP)
       WRITE(*,*)'PROCESSOR',NP,'NCELLS',NC,(FLOAT(NC)/FNTEVEN)
      END DO
      RETURN
      END
```


Other Mapping Techniques

- **The mapping of blocks, nodes, or cells onto processors is a subject of research that has been ongoing since the advent of parallel computing**
- **Several researchers have developed open-source libraries of static, graph-partitioning schemes:**
  - Chaco: B. Hendrickson and R. Leland (Sandia National Lab)
    http://www.cs.sandia.gov/CRF/chac.html
  - Metis/Parmetis:G. Karypis and V. Kumar (University of Minnesota)
    http://glaros.dtc.umn.edu/gkhome/views/metis/
  - Many others (do a web-search under “graph partitioning”)

Chaco

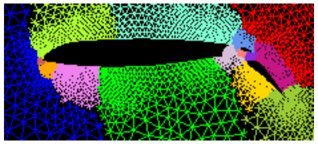
- **Chaco is a graph partitioning system**
  - From Sandia National Lab (not sure of the availability of source-code)
  - Allows for recursive application of several methods for finding small edge separators in weighted graphs.
    - “These methods include inertial, spectral, Kernighan-Lin and multilevel methods in addition to several simpler strategies.”*
    - “Each of these approaches can be used to partition the graph into two, four or eight pieces at each level of recursion. In addition, the Kernighan-Lin (optimization) method can be used to improve partitions generated by any of the other algorithms.”*



*From Chaco users’ manual

Jostle

- **Jostle was an open-source library for static and dynamic graph partitioning that are both serial and parallel (similar to Metis/Parmetis)(however, no longer available)**
  - Combination of a graph contraction algorithm (similar to grid-sequencing) with a local optimization method which refines the partition at each graph level
  - Kernighan-Lin partition optimization which incorporates load-balancing
  - Evolutionary search algorithms



Metis/Parmetis

- **Metis and Parmetis are open-source codes/libraries that also perform graph partitioning**
- **I have downloaded Metis and Parmetis and put it in the “Additional Material” folder in smartsite for your information**
- **Metis and Parmetis (parallel version of Metis) can be run as a separate program or has subroutines that may be linked and called from your program (Metislib).**
- **Metis may be used to partition structured- or unstructured nodes, cells, or blocks**
  - Blocks may be treated as cells with weighted nodes and/or edges
  - Minimizes either the number of edges that straddle partitions (edgecut) or the total communication volume (totalv)

Metis/Parmetis

- **Multi-Constraint Partitioning**
  - Each vertex or edge (of a cell or block) has a vector of weights of size *m*
  - “The objective of the partitioning algorithm is to minimize the edgecut subject to the constraints that each one of the *m* weights is equally distributed among the domains.”*
- **Minimizing the Total Communication Volume**
  - Objective of traditional graph partitioning is to compute a balanced k-way partitioning such that the number of edges (or the sum of their weights) that straddle different partitions is minimized. The objective of minimizing the edgecut is only an approximation of the true communication cost.
  - Metis can also directly minimize the communication cost as defined by the total communication volume.

Metis/Parmetis

- **Minimizing the Maximum Connectivity of the Subdomains:**
  - Communication costs generally depends on:
    - The total communication volume
    - The maximum amount of data that any particular processor needs to send and receive
    - The number of messages a processor needs to send and receive
  - Metis attempts to reduce all three factors
- **Reducing the Number of Non-Contiguous Subdomains**
  - A k-way partitioning of a continuous graph can often lead to some sub-domains being assigned non-contiguous portions of the graph (i.e. certain domains are broken needlessly)
  - Metis attempts to eliminate these non-contiguous subdomains

Homework 6 (cont)

- Read Chapter 5 of Introduction to Parallel Computing by Grama et. al.