Application of the Multi-level Parallelism (MLP) Software to a Finite Element Groundwater Program Using Iterative Solvers with Comparison to MPI

Fred Tracy

Engineer Research and Development Center
Information Technology Laboratory
Major Shared Resource Center
Vicksburg, MS, USA 39180

Abstract. The purpose of this paper is to give the results of the performance evaluation of the Multi-Level Parallelism (MLP) software versus MPI using the groundwater program FEMWATER on a remediation of a military site where iterative solvers are employed to solve the system of nonlinear equations. A one-to-one correspondence in functionality between MPI and MLP was maintained for all parallel operations so the performance comparisons would be consistent. An unstructured mesh application is one of the most difficult parallel applications, so this represents a good test. In this study, MLP did better in general on 64 PEs or less, but MPI proved more scalable as it did as good or better when using 128 PEs.

1 Introduction

Users of large shared-memory architectures on high performance computers are often challenged to find a method of parallel programming that is both efficient and easy to use. On shared-memory machines with large numbers of processors, the question of the "best" parallel method for a given problem is a difficult one. Recently, the Message Passing Interface (MPI) has been combined with OpenMP threads to make a dual-level or mixed-mode parallel programming paradigm [1]. While OpenMP is rather simple to apply to an existing code, it has inherent limitations with respect to efficient parallelization for larger numbers of processors. If programmed properly, MPI can create codes that scale to very large numbers of processors, but it is typically rather difficult to implement.

The Multi-Level Parallelism (MLP) software [2] utilizes a new method that takes advantage of large shared-memory SGI architectures, and excellent performance on NASA Ames applications has been reported [3]. An independent evaluation of MLP [4] gave inconclusive results on the performance of MLP and the NASA Ames computational fluid dynamics code Overflow because the MPI version and the MLP version of Overflow that were used in the evaluation had different algorithms for partitioning the work. Therefore, a more detailed study is warranted.

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 725–735, 2003.

1.1 Purpose

The purpose of this paper is to give the results of the performance evaluation of MLP versus MPI using the groundwater program FEMWATER [5] on a remediation of a military site [6] where iterative solvers are employed to solve the system of nonlinear equations. A one-to-one correspondence in functionality between MPI and MLP was maintained for all parallel operations so the performance comparisons would be consistent. This was done, for instance, by replacing a call to an MPI reduction routine to get the global maximum value by a call to an equivalent MLP subroutine (described below). Another example also described below is the updating of ghost nodes where an equivalent subroutine was written in MLP. An unstructured mesh application is one of the most difficult parallel applications, so this represents a good test.

1.2 The MLP Parallel Programming Paradigm

Using MLP, the problem is explicitly partitioned among processing elements (PEs) as in MPI. However, rather than using sends, receives, broadcasts, reductions, etc., as in MPI, MLP communicates data among PEs by using shared variables as in OpenMP. As in MPI, combining OpenMP threads with forked processes is very natural with MLP, which, in fact, is indicated in its name.

2 Description of the Application

Figure 1 illustrates a typical top view of a 3-D finite element mesh for a remediation study. Several layers are used to model the soil layers underneath this surface. The mesh used for this comparison has 102,996 nodes and 187,902 3-D prism elements. The number of PEs used in this study starts at eight and is repeatedly doubled to 16, 32, 64, and 128 with the number of elements also being doubled each time the number of PEs is doubled.

3 Flow Equations

Pressure head in FEMWATER is modeled by applying conservation of mass to obtain,

$$\frac{\rho}{\rho_0} F \frac{\partial h}{\partial t} = \bigtriangledown \cdot \left[\mathbf{K} \cdot \left(\bigtriangledown \mathbf{h} + \frac{\rho}{\rho_\mathbf{0}} \bigtriangledown \mathbf{z} \right) \right] + \sum_{m=1}^{N_{ss}} \frac{\rho_m^*}{\rho_0} Q_m \delta \left(\mathbf{r} - \mathbf{r_m} \right) \ , \tag{1}$$

$$h = \frac{p}{\rho_0 g} \ , \tag{2}$$

$$F = n \frac{dS}{dh} + S\alpha' + \theta\beta' \ , \tag{3}$$

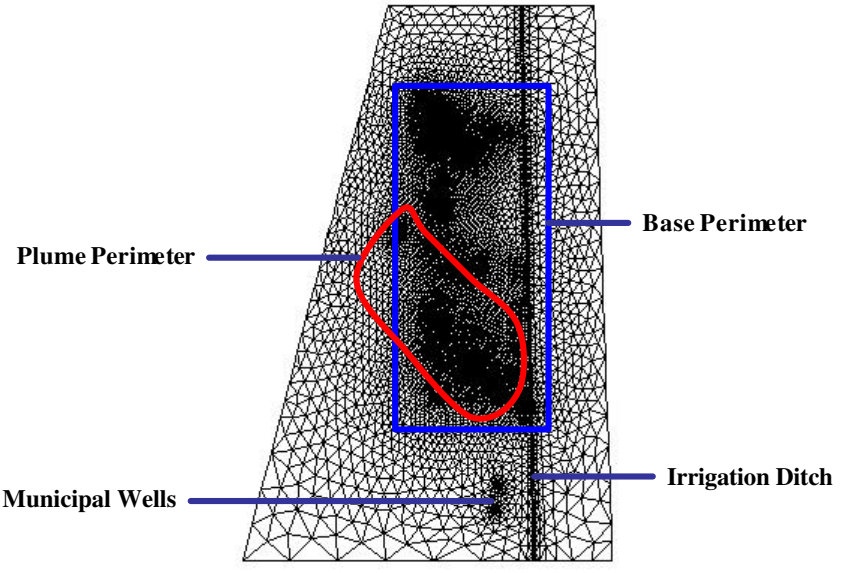

Fig. 1. This figure illustrates a typical top view of a 3-D finite mesh for a remediation study. Several layers are used to model the soil layers underneath the surface. The mesh used for this comparison has 102,996 nodes and 187,902 3-D prism elements

$$\alpha' = \rho_0 g \alpha \ , \tag{4}$$

$$\beta' = \rho_0 g \beta \ . \tag{5}$$

where α is the compressibility of the soil medium, β is the compressibility of water, δ is the Dirac delta function, g is the acceleration due to gravity, h is the pressure head, $\mathbf{K}$ is the hydraulic conductivity tensor, n is the porosity, N_{ss} is the number of source/sink nodes for flow, Q_m is the quantity of flow at the m^{th} source/sink node, p is the pressure, $\mathbf{r}$ is a vector from the origin to an (x, y, z) point in space, ρ is the density with contaminant, ρ_0 is the density without contaminant, ρ_m^* is the density of the m^{th} source/sink, $\mathbf{r_m}$ is the location of the m^{th} source/sink node, S is the saturation, t is the time, and θ is the moisture content.

The Galerkin finite element method is then applied to obtain

$$\mathbf{M^{n+1}} \left(\mathbf{h^{n+1}} - \mathbf{h^n} \right) + \triangle \mathbf{t} \mathbf{K^{n+1}} \mathbf{h^{n+1}} = \triangle \mathbf{t} \mathbf{Q}^{'\mathbf{n}} \ . \tag{6}$$

for the $(n+1)^{th}$ time-step. Here $\mathbf{M}$ is the mass matrix, $\mathbf{K}$ is the stiffness matrix, and $\mathbf{Q'}$ is a collection of flow type terms for the right-hand side, and $\triangle t$

is the time increment. Equation (6) is the resulting system of nonlinear equations that is solved, where both **M** and **K** are symmetric.

4 Parallel Paradigm

The finite element mesh is first partitioned using METIS [7]. The ghost cells as illustrated in Fig. 2 are updated using either MPI or MLP. Border elements are kept by both PEs and owned by the PE that owns the first node of the element.

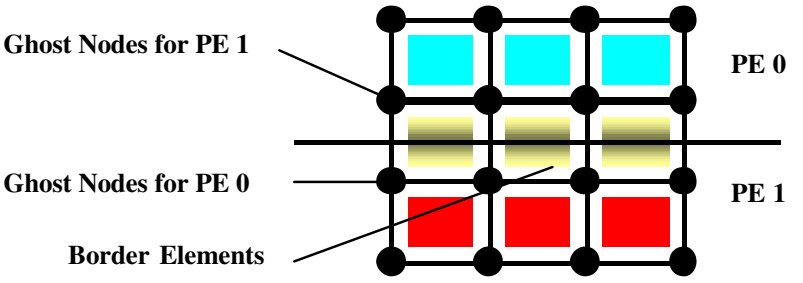

Fig. 2. The ghost cells are updated using either MPI or MLP. Border elements are kept by both PEs and owned by the PE that owns the first node of the element

5 Solvers Tested

Equation (6) is solved using a Picard iteration resulting in a symmetric, positive-definite linear system of equations of the typical form,

$$\mathbf{Ax} = \mathbf{b} \,. \tag{7}$$

The first solver tested consists of a forward relaxation step described by

$$\mathbf{x^{new}} = \mathbf{x^{old}} + \omega \left(\mathbf{D} + \mathbf{L}\right)^{-1} \left(\mathbf{b} - \mathbf{Ax^{old}}\right) \quad \mathbf{0} < \omega \leq \mathbf{2} \,, \tag{8}$$

followed by a backward relaxation step,

$$\mathbf{x^{new}} = \mathbf{x^{old}} + \omega \left(\mathbf{D} + \mathbf{U}\right)^{-1} \left(\mathbf{b} - \mathbf{Ax^{old}}\right) \quad \mathbf{0} < \omega \leq \mathbf{2} \,. \tag{9}$$

where **D** is the diagonal of **A**, **L** is the lower part of A, **U** is the upper part of **A**, and ω is the relaxation factor. In the parallel version, the owned nodes do the relaxation in parallel with matrix elements in **L** (forward step) and **U** (backward step) set to zeroes anywhere beyond the owned nodes and the ghost nodes. The ghost nodes are updated after the forward step and again after the backward step.

The second solver tested is a conjugate gradient solver with diagonal and incomplete LU preconditioners. The ILU preconditioner [8] is of the form,

$$\tilde{\mathbf{A}} = (\mathbf{D} + \omega\mathbf{L})\,\mathbf{D}^{-1}\,(\mathbf{D} + \omega\mathbf{U}) \quad \mathbf{0} \leq \omega \leq \mathbf{1}\,, \tag{10}$$

It was implemented to minimize communication by setting to zero all terms in $\mathbf{L}$ and $\mathbf{U}$ that were beyond owned and ghost nodes of the PE. Note that with $\omega = 0$, the preconditioner becomes $\mathbf{D}$.

6 MLP and MPI Programming Details

The MLP and MPI programming details will now be presented in detail.

6.1 Initialization

While MPI is an extensive library, MLP has relatively few routines. Therefore, more is required when using MLP to set up. With MLP, rather than using sends, receives, broadcasts, reductions, etc., as in MPI, shared variables are used to communicate information as in OpenMP. The computer code given below shows the setup process as was done in FEMWATER.

MLP Setup Process for FEMWATER

```
 implicit real * 8 (a-h, o-z)
 parameter (ighnmx = 6113, npmx = 16)
 dimension bufv8(ighnmx, npmx), bufs8(npmx), ibufs(npmx)
 common / mlp_dat / ipt1, ipt2, ipt3
 dimension numcpu(npmx)
c   Establish pointers.
 pointer (ipt1, bufv8)
 pointer (ipt2, bufs8)
 pointer (ipt3, ibufs)
 integer * 8 isizes(3), ipoint(3), numvar
c   Get memory and link with pointers.
 numvar = 3
 isizes(1) = ighnmx * npmx * 8
 isizes(2) = npmx * 8
 isizes(3) = npmx * 4
 call mlp_getmem (numvar, isizes, ipoint)
 ipt1 = ipoint(1)
 ipt2 = ipoint(2)
 ipt3 = ipoint(3)
c   Set the number of processes.
 noproc = npmx
c   Pin-to-node option.
 npinit = 1
```

```
c   Choose one thread per process.
 do i = 1, noproc
   numcpu(i) = 1
 end do
c   Fork processes.
 call mlp_forkit (noproc, myid, numcpu, npinit)
```

First, the general-purpose shared variables of `bufv8`, `bufs8`, and `ibufs` are defined to keep a real vector for each PE, a real scalar for each PE, and an integer scalar for each PE, respectively. The pointer variables are then used to associate the value of the pointer to where the variable resides in memory as determined by the call to `mlp_getmem`. The pointers can be placed in a common block, but the actual variables cannot, as their memory locations are already set in `mlp_getmem`.

In the example above, the number of processes is set to 16, and the number of threads per process is set to 1 (OpenMP directives are not inserted). Note that it is easy to set the number of threads to a different value for each process. Threads can also migrate from their assigned node (the SGI O3K has four PEs per node), but experience indicates that performance improves when using multiple threads per process if the threads are pinned to their respective nodes as done by `ipinit` = 1. The call to `mlp_forkit` finishes the initialization process by forking the processes.

6.2 Reduction

A typical reduction in FEMWATER using MPI is given by

Reduction Using MPI

```
 call MPI_ALLREDUCE (res, resg, 1, MPI_REAL8, MPI_MAX,
&  MPI_COMM_WORLD, ieror)
```

where the maximum value of `res` over all the PEs is placed into `resg`. The program code given below shows how this is done using MLP.

Reduction Using MLP

```
c   Put data in the shared variable.
 bufs8(myid) = res
c   Make sure all PEs are through.
 call mlp_barrier (myid, noproc)
c   Have all PEs compute the maximum value.
 gmax = -1.0d30
 do i = 1, noproc
   gmax = dmax1 (bufs8(i), gmax)
 end do
 resg = gmax
```

Here, `myid` goes from 1 to 16, and each PE places its value of `res` into the `bufs8` variable. Adding the `mlp_barrier` call ensures that all the PEs have finished. Finally, the maximum value is manually computed and stored in `resg`.

6.3 Updating Ghost Nodes

Most of the communication in FEMWATER is done in the solver when updating the ghost nodes. The send loop using MPI is as follows:

MPI Send Loop

```
 do i = 1, noproc
   num = nodgh(1, i)
   if (num .ne. 0) then
     jfn = num + 1
     do j = 2, jfn
       jloc = nodgh(j, i)
       buff(j - 1) = v(jloc)
     end do
     itag = 100
     call MPI_SEND (buff, num, MPI_REAL8, i - 1, itag,
&      MPI_COMM_WORLD, ierror)
   end if
 end do
```

The store data loop using MLP is

MLP Store Data Loop

```
 ibufs(myid) = 0
 iplace = 0
 do i = 1, noproc
   num = nodgh(1, i)
   if (num .ne. 0) then
     ibufs(myid) = ibufs(myid) + 1
     iplace = iplace + 1
     bufv8(iplace, myid) = i
     iplace = iplace + 1
     bufv8(iplace, myid) = num
     jfn = num + 1
     do j = 2, jfn
       iplace = iplace + 1
       jloc = nodgh(j, i)
       bufv8(iplace, myid) = v(jloc)
     end do
   end if
 end do
```

`nodgh` is an array that contains the number of nodes and the local node numbers whose values are to be sent to different PEs. The MLP version stores the number of messages in `ibufs` and the actual data of the messages in the `bufv8` array. The receive loop (actually done before the send loop) using MPI is the following:

MPI Receive Loop

```
 do i = 1, noproc
   num = numngh(i)
   if (num .ne. 0) then
     itag = 100
     nst = nstngh(i)
     call MPI_IRECV (v(nst), num, MPI_REAL8, i - 1,
&      itag, MPI_COMM_WORLD, ireq(i), ierror)
   end if
 end do
```

The wait loop for MPI to complete the ghost cell update is done by

MPI Wait Loop

```
 do i = 1, noproc
   if (numngh(i) .ne. 0) then
     call MPI_WAIT (ireq(i), istat, ierror)
   end if
 end do
```

`numgh` is the number of ghost cells to be updated from the various PEs, and `nst` is where the ghost cells start in the local `v` vector. Finally, the retrieve data loop for MLP is the following:

MLP Retrieve Data Loop

```
 do i = 1, noproc
   nummes = ibufs(i)
   if (nummes .ne. 0) then
     iplace = 0
     do k = 1, nummes
       iplace = iplace + 1
       idest = bufv8(iplace, i)
       iplace = iplace + 1
       num = bufv8(iplace, i)
       if (idest .eq. myid) then
         nst = nstngh(i)
         do j = 1, num
           v(j + nst - 1) = bufv8(iplace + j, i)
         end do
```

```
      end if
      iplace = iplace + num
    end do
  end if
end do
```

Synchronization is achieved by placing a call to `mlp_barrier` between the store and retrieve loops.

7 Performance Results

Performance tests using both the MPI and MLP version of FEMWATER were conducted on the SGI Origin 3800 located at the U.S. Army Engineer Research and Development Center (ERDC) Major Shared Resource Center (MSRC), Vicksburg, MS, for the remediation test problem using three different iterative solvers. Table 1 summarizes the results. The iterative linear solvers used were (1) the relaxation solver with one linear iteration being a forward loop, a ghost node update, a backward loop, and a second ghost node update, (2) a CG solver with the diagonal preconditioner, and (3) a CG solver using the ILU preconditioner with $\omega = 1$. Five hundred nonlinear iterations with 20 linear iterations each were done in each case. Both CG solves converged much quicker than the relaxation solver, but since this is a performance test of MPI versus MLP, the same number of linear iterations (10,000) was done in each case. In this series of tests, no OpenMP directives were added. The original problem was first run with 8 PEs. Next, the number of PEs was doubled along with the number of elements until 128 PE runs were done. These tests were run in a very busy production environment on the O3K, so running times varied as much as 10%. The best times from two runs in each case were used for Table 1.

8 Conclusions

The MLP results were generally better than the MPI results for PEs 8-64, and the MPI results were as good or better when 128 PEs were used. The percentage differences were always less than 10%, which means that no significant advantage was achieved with either parallel paradigm. The number of source lines of code (SLOC) not counting comments for the single subroutine that does the ghost cell update is 35 for MPI and 52 for MLP, so each is rather simple to implement, especially since the tedious work of determining where the data are to be sent and received has already been done. However, the edge goes to MPI for simplicity. More SLOC are required in the initialization and reduction for MLP, but the MPI libraries are much more extensive and thus already have initialization and reduction routines supplied.

For FEMWATER, a one-to-one correspondence implementation from MPI to MLP does not yield much benefit. However, if the shared variables could be used in a clever way to avoid all the partitioning and preparation, MLP still

Table 1. FREMWATER running times (sec)

PEs	8	16	32	64	128
Nodes	102,996	197,409	386,235	763,887	1,519,191
Elements	187,902	375,804	751,608	1,503,216	3,006,432
Solver	Relaxation	Relaxation	Relaxation	Relaxation	Relaxation
MPI	506	533	569	615	612
MLP	495	531	558	590	675
Solver	CG	CG	CG	CG	CG
Preconditioner	Diagonal	Diagonal	Diagonal	Diagonal	Diagonal
MPI	464	484	505	514	582
MLP	437	473	492	524	619
Solver	CG	CG	CG	CG	CG
Preconditioner	ILU	ILU	ILU	ILU	ILU
MPI	601	637	639	668	708
MLP	581	613	639	656	708

could have merit. Further research is needed here, as well as investigating the use of multiple OpenMP threads for each process.

Acknowledgment. This work was supported in part by a grant of computer time from the DoD High Performance Computing Modernization Program at the ERDC MSRC.

References

1. Fahey, R, and Smith, J.: STWAVE: A Case Study in Dual-Level Parallelism. ERSC MSRC Technical Report 01-28, Vicksburg, MS (2001)
2. Taft, J.R.: MLP Version 2.1 (computer program). Sienna Software, Inc., 1105 Terminal Way, Ste 202, Reno, NV (2002)
3. Taft, J.R.: Overflow Gets Excellent Results on SGI Origin 2000. NAS News, Vol. 3, No. 1, NASA Ames Research Center, Moffett Field, CA (1998)
4. Wornom, S.F., Tracy, F.T., Duffy, D.Q., and Alter, R.W.: A Performance Evaluation of the Multi-Level Parallelism (MLP) Software with MPI and OpenMP. DoD HPC Users Group Conference Proceedings, Austin, TX (2002)
5. Lin, H.J., Richards, D.R., Talbot, C.A., Yeh, G.T., Cheng, J.R., Cheng, H.P., and Jones, N.L.: FEMWATER: A Three-Dimensional Finite Element Computer Model for Simulating Density-Dependent Flow and Transport in Variably Saturated Media. Technical Report CHL-97-12, U.S. Army Engineer Research and Development Center (ERDC), Vicksburg, MS (1997)

6. Tracy, F.T., Talbot, C.A., Holland, J.P., Turnbull, S.J., McGehee, T.L., and Donnell, B.P.: The Application of the Parallelized Groundwater Model FEMWATER to a Deep Mine Project and the Remediation of a Large Military Site. DoD HPC Users Group Conference Proceedings, Monterey, CA (1999)
7. Karypis, G.: METIS (computer program). `http://www.users.cs.umn.edu/~karypis/metis/`, University of Minnesota, Minneapolis, MN (2002)
8. Dongara, J.J., Sorensen, D.C., and van der Vorst, H.A.: Numerical Linear Algebra for High-Performance Computers, SIAM, Philadelphia (1998), 203