Týden 4

Programování zasíláním zpráv (MP)

- programování v MPI
- počítání na svazcích stanic

Proč potřebujeme komunikátory? - pokrač.

Nelze zajistit různost příznaků:
- neznáme příznaky v knihovních podprogramech, nebo
- někdy se používá **MPI_Any_tag**.

Řešení v MPI jsou komunikátory. I když jde třeba o stejné skupiny procesů, systém jim přidělí různý kontext (komunikátor, tj. číslo).
Komunikace v různých komunikátorech jsou odděleny a každá kolektivní komunikace je oddělena od každé p2p (dvoubodové) i ve stejném komunikátoru.

Intrakomunikátor - komunikace v rámci skupiny
Interkomunikátor - pro komunikaci mezi skupinami (MPI-2)

Anatomie příkazu send v MPI

MPI_Send(&N, 1, MPI_INT, i, i, MPI_COMM_WORLD)

- jméno podprogramu
- Adresa zprávy
- délka (počet zpráv)
- Identifikace cílového procesu
- příznak zprávy
- komunikátor

Nelze uvádět délku v byte

Datový typ zprávy; může být jiný v různých zemích, např. znak je 8 bitů (Evropa) nebo 16 bitů (Čína)

Příklad blokující komunikace v MPI

Poslat int *x* z procesu 0 do procesu 1:

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /*najdi svuj index*/
if (myrank == 0) {
   int x;
   MPI_Send(&x, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD);
}  else if (myrank == 1) {
   int x;
   MPI_Recv(&x, 1, MPI_INT, 0, msgtag, MPI_COMM_WORLD, status);
}
```

komu

od koho

stav po operaci

Standardní send, nečeká na receive, buffer není v MPI definován; pokud je k dispozici, send může skončit dříve než se dojde na recv.

Příklad neblokující komunikace v MPI

Poslat int x z procesu 0 do procesu 1 a dovolit procesu 0 pokračovat:

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0) {
   int x;
   MPI_Isend(&x, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD, req1);
   compute();
   MPI_Wait(req1, status);
} else if (myrank == 1) {
   int x;
   MPI_Recv(&x, 0, MPI_INT, 1, msgtag, MPI_COMM_WORLD, status);
}
```

parametr „popisovač“

MPI program typu SPMD pro $\Sigma f(i)$

```
#include "mpi.h"
int f(i)  int i; { ... }
main(argc, argv)
int argc; char*  argv[];
{   int i, tmp, sum=0, group_size, my_rank, N;
    MPI_Init (&argc, &argv);
    MPI_Comm_size( MPI_COMM_WORLD, &group_size);
   MPI_Comm_rank( MPI_COMM_WORLD, &my_rank);
   if ( my_rank == 0 )  {
      printf("Enter N: "); scanf(" % d ", &N);
      for ( i=1; i<group_size; i++)    /*one-to-all broadcast N */
S1:      MPI_Send(&N, 1, MPI_INT, i, i, MPI_COMM_WORLD);
      for (i=my_rank; i<N; i = i +group_size)
         sum = sum + f(i);   /*f(0)+f(P)+f(2P)+...   */
```

MPI program, pokrač.

```
      for ( i=1; i<group_size; i++)  {   /* all-to-one gather */
S2:       MPI_Recv(&tmp, 1, MPI_INT, i, i, MPI_COMM_WORLD, &status);
          sum = sum + tmp; }
      printf ("\n The result = %d", sum);
    }
    else {        /* if my_rank != 0 */
S3:   MPI_Recv(&N, 1, MPI_INT, 0, i, MPI_COMM_WORLD, &status);
      for (i=my_rank; i<N; i=i+group_size)
        sum = sum + f(i);    /*f(k)+f(k+P)+f(k+2P)+... k=my_rank */

S4:   MPI_Send(&sum, 1, MPI_INT, 0, i, MPI_COMM_WORLD);
    }
    MPI_Finalize( );
}
```

Tvorba nových komunikátorů

Kromě duplikace též vyloučením podskupiny procesů, inkluzí, použitím množinových operací ...

- z podmnožiny procesů subgroup spojených s existujícím komunikátorem
  MPI_Comm_create (oldcomm, subgroup, &newcomm)
- z disjunktních podskupin procesů podle klíče
  MPI_Comm_split (oldcomm, color, key, &SplitWorld)

příkaz musí být obsažen ve všech procesech starého komunikátoru.

Příklad: Nechť má oldcomm 10 procesů; použijeme-li
color = my_rank% 3 a key = my_rank/3 , dostaneme

index v oldcomm			0	1	2	3	4	5	6	7	8	9
color			0	1	2	0	1	2	0	1	2	0
key			0	0	0	1	1	1	2	2	2	3
index ve SplitWorld		(color=0)	0			1			2			3
index ve SplitWorld		(color=1)		0			1			2		
index ve SplitWorld		(color=2)			0			1			2	

Virtuální topologie v MPI-1

- Komunikátor může být spojen s topologií
- 2 virtuální topologie: kartézská a grafová
- MPI_Cart_create() vrací nový komunikátor s kartézskou strukturou; lze získat nový index procesu nebo jeho kartézské souřadnice;
- MPI_Graph_create() vrací nový komunikátor, v němž jsou procesy uspořádány do obecného grafu.

- Virtuální topologii lze optimálně mapovat na fyzické procesory ležící pod ní - povoluje se ve funkci create.
- Virtuální topologie umožňuje praktičtější schéma adresace procesů.

Datové typy v MPI

- Předdefinované:

MPI_BYTE
MPI_CHAR
MPI_DOUBLE
MPI_FLOAT
MPI_INT
MPI_LONG
MPI LONG_DOUBLE
MPI_UNSIGNED
(CHAR, SHORT, LONG)
MPI_PACKED

- Odvozené (příklad):

EvenElements = EE
double A[100];
MPI_Data_type EE;
...
MPI_Type_vector(50, 1, 2, MPI_DOUBLE, &EE);
MPI_Type_commit(&EE) ;
MPI_Send(A, 1, EE, dst, ...)

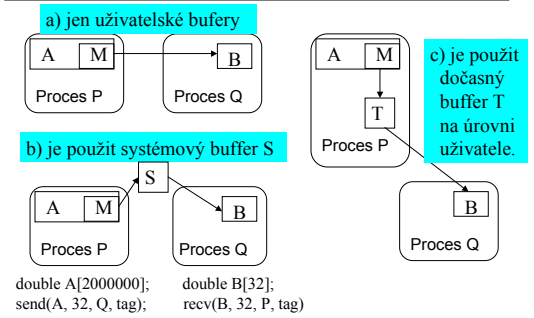
(count, blocklength, stride, oldtype, &newtype)

Různé operace Send/Receive v MPI

Příkaz MPI	Blokující	Neblokující
Standardní send	MPI_Send	MPI_Isend
Synchronní send	MPI_Ssend	MPI_Issend
Buffered Send	MPI_Bsend	MPI_Ibsend
Ready Send	MPI_Rsend	MPI_Irsend
Receive	MPI_Recv	MPI_Irecv
Test hotovo?	MPI_Wait	MPI_Test

Buffered Mode: Je nutné specifikovat prostor bufferu pomocí MPI_Buffer_attach() - odstranění pomocí MPI_Buffer_detach().
Synchronous Mode: Send a receive může začít jeden před druhým, ale mohou skončit pouze spolu.
Ready Mode : Send může začít jen když se dosáhlo odpovídající receive, jinak chyba. Používat opatrně.

Tři typy buferů zpráv v MPI



double A[2000000];
send(A, 32, Q, tag);

double B[32];
recv(B, 32, P, tag)

Deadlock při komunikaci v MPI

- může vzniknout, když dva procesy posílají a přijímají zprávy jeden od druhého;
- deadlok vznikne když oba procesy provedou synchronní send, nebo blokující send bez dostatečně velkého buferu;
- žádný proces se ze send nevrátí, protože čeká na odpovídající recv, kterého se nikdy nedosáhne;
- program SPMD je nutno upravit tak, aby jeden partner měl sekvenci send, recv a druhý partner recv, send, např. sudé a liché procesy;
- MPI také poskytuje kombinované operace bez deadlocku MPI_Sendrecv() a MPI_Sendrecv_replace()

P_{i-1} P_i P_{i+1}

sendrecv(P_i); ⟷ sendrecv(P_{i-1});

sendrecv(P_{i+1}); ⟷ sendrecv(P_i);

Synchronizace v MPI

- zajištění následnosti použitím blokujícího Receive:

```
f1( ) /*call the function f1 */                task 0
MPI_Send(sbuf,1,MPI_int, 5, 99 , comm_space)

…                                              task 5
MPI_Recv (rbuf,1,MPI_int, 0, 99 , comm_space, &status)
f2( ) /*call the function f2 */
```

f1( ) skončí před začátkem f2( )

- synchronizace při komunikaci (rendezvous): MPI_Ssend( ) ⟶ MPI_Recv( )

- MPI_Barrier(comm_space)

Kolektivní operace v MPI

- zahrnují množinu procesů definovaných komunikátorem
- jsou blokující
- příznaky zpráv se zde nepoužívají
- všechny procesy musí obsahovat volání kolektivní operace.

Globální kombinace (předdefinovaná nebo definovaná uživatelem) do root
MPI_Reduce(sbuf, rbuf, n, data_type, op, rt, communicator)

Redukce mnoha do mnoha, operace Prefix scan
MPI_Allreduce(sbuf, rbuf, n, data_type, op, communicator)
MPI_Scan(sbuf, rbuf, n, data_type, op, communicator)

Operace přesunující data
MPI_Bcast(buffer, n, data_type, root, communicator)
MPI_Scatter(sbuf, n, stype, rbuf, m, rtype, rt, communicator)
MPI_Gather(sbuf, n, stype, rbuf, m, rtype, rt, communicator)
MPI_Alltoall( )

MPI_SUM
MPI_PROD
MPI_MIN
MPI_MAX
MPI_LAND
MPI_LXOR
MPI_BOR
a další

Kolektivní operace v MPI - příklad

Posbírání hodnot ze skupiny procesů do procesu 0, včetně procesu 0, použitím dynamicky alokované paměti v procesu root

```
int data[10];                /*data která se mají posbirat z procesů*/
. . .
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);    /* najdi rank */
if (myrank == 0) {                /*najdi velikost skupiny a alokuj paměť */
    MPI_Comm_size(MPI_COMM_WORLD, &grp_size);
    buf = (int  *)malloc(grp_size*10*sizeof(int));
}
MPI_Gather(data, 10, MPI_INT, buf, grp_size*10, MPI_INT, 0,
            MPI_COMM_WORLD);
```

kdo sbírá

Příklad - sčítání

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
#define MAXSIZE 1000
void main(int argc, char *argv)
{
   int myid, numprocs;
   int data[MAXSIZE], i, x, low, high, myresult, result;
   char fn[255];
   char *fp;
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   if (myid == 0) { /* Open input file and initialize data */
      strcpy(fn,getenv("HOME"));
      strcat(fn,"/MPI/rand_data.txt");
      if ((fp = fopen(fn,"r")) == NULL) {
          printf("Can't open the input file: %s\n\n", fn);
          exit(1);
      }
      for(i = 0; i < MAXSIZE; i++) fscanf(fp,"%d", &data[i]);
   }
```

Příklad - pokrač.

```
/* broadcast data */
   MPI_Bcast(data, MAXSIZE, MPI_INT, 0, MPI_COMM_WORLD);
/* Add my portion of data */
   x = n/nproc;
   low = myid * x;
   high = low + x;
   for(i = low; i < high; i++)
   myresult += data[i];
   printf("I got %d from %d\n", myresult, myid);
/* Compute global sum */
   MPI_Reduce(&myresult, &result, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
   if (myid == 0) printf("The sum is %d.\n", result);
   MPI_Finalize();
}
```

Počítání na svazcích stanic - odkazy

Informační centrum:
http://www.csse.monash.edu.au/~rajkumar/cluster/

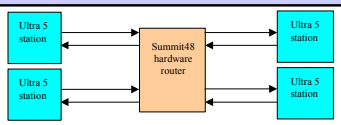
IEEE Computer Society Task Force on Cluster Computing:
http://www.ieeetfcc.org/

Clusters@TOP500: (Top 10 clusters)
http://clusters.top500.org/

Poslední vývoj ve svazcích

- Beowulf - svazek Linuxových pracovních stanic nebo PC
- svazky SMP, thread-safe MPI je rozšíření MPI pro práci s vlákny a sdílenou pamětí (z OpenMP lze volat funkce z knihovny MPI).
- V poslední době se i superpočítače koncipují jako svazky SMP.
- Meta-computing, Hyper-computing, Remote Message Passing RMP, grid computing - účinnost není důležitá, hlavní je udělat a dokončit velký objem výpočtů.

Svazek pracovních stanic (FIT)



Stanice Ultra 5: CPU: UltraSPARC II @ 270 MHz + linková karta.
HW směrovač: Summit48 (firmy Extreme Networks) s 48 bránami Ethernet na 100 Mbit/s a dvě Ethernetové brány na 1 Gbit/s.
Směrovač Summit48 má celkově propustnost 17.5 Gbit/s, neblokující strukturu přepínačů a přepravuje 10.1 milionu paketů za sekundu.

Zpoždění HW směrovače : pod 10 µs a dá se zanedbat.
Zpoždění SW : 160 µs pro send a 260 µs pro receive při hodinovém kmitočtu 270 MHz.

Svazky PC s Linuxem na FIT

Na 65 stanicích s Linuxem je nainstalovan balík LAM (Local Area Multiprocesor, implementace MPI).
Jedná se o tyto počítače;

PC00 - 09	PCA0 - A9	PCB0 - B9
PCC0 - C9	PCD0 - D9	PCE0 - E1
PC50 - 61		

+ server merlin.dcse.fee.vutbr.cz

Systém lineárních rovnic na svazku stanic

Gauss - Jordanova metoda řešení jedním průchodem

```
for i = 1 to n do                          {číslo pivotního řádku}
      for j = 1 to n, j <> i do            (číslo řádku}
        begin
          c_j := a_ji / a_ii               {násobící činitel}
          b_j := b_j - c_j b_i
          for k = i+1 to n do              {číslo prvku}
            a_jk := a_jk - c_j a_ik        {řádek - činitel * pivotní řadek}
          end
    for i = 1 to n do
        x_i := b_i / a_ii                  {řešení}
```

Paralelizace:
Každý procesor n/P rovnic; pak jeden procesor po druhém řídí postup - posílá pivotní řádek opakovaně všem ostatním (vystačíme s OAB).

Paralelní řešení systému lineárních rovnic

Poznámky:
- délka pivotního řádku se systematicky snižuje
- jediný typ komunikace: broadcast
- dvě možnosti, počáteční distribuce dat je zahrnuta do času řešení nebo nikoliv
- rozesílání pivotního řádku je překryto s počítáním, v němž se používá předchozí pivotní řádek (SW řetěz)

Řešení po skončení kroku i:

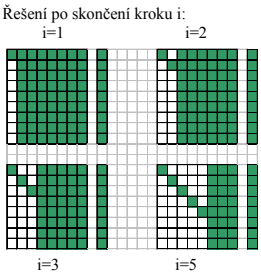


Schéma programu (SPMD)

```
# P fází výpočtu, všechny procesory se střídají
for [i=0 to P-1]
   if (i=myid) {  #  jsem v této fázi řídící procesor
      for [k=0 to n/P-1]
          # rozhlašuji pivotní řádky,
          # pracuji na svém bloku rovnic
    }else{        # i ! = myid
      for [k=0 to n/P-1]
          # jsem v této fázi řadový procesor ; přijímám
          # pivotní řádky a pracuji na svém bloku rovnic
    }
```

Příště:
Bariéra a synchronní iterační výpočty.
Bezpečné zasílání zpráv v MPI.