MPI (Message Passing Interface)

– rozhraní pro (explicitní) programování paralelních aplikací na jednom i více počítačích s distribuovanou pamětí
– startuje několik procesů (každý s nezávislou pamětí) zasílajících si za chodu zprávy (data)
– zdrojový kód procesů obvykle totožný (SPMD, single program multiple data), možné i schéma master-slave (MPMD)
– realizace: podprogramy, funkce, proměnné a konstanty, proměnné prostředí
– překlad pomocí wrapperů (př. mpif90) zajišťujících připojení knihoven, spouštění pomocí agenta (mpirun/mpiexec)
– http://www.mpi-forum.org, dokumentace MPI: A Message-Passing Interface Standard ver. 2.2 (PDF 647 stran)
– 1994 standard MPI-1 (2008: ver. 1.3), 1996 MPI-2 (2009: ver. 2.2)
– implementace: knihovny (včetně zdrojových kódů) Open MPI (ver. 1.4 s MPI-2), MPICH aj. pro Fortran, C/C++ aj.

Součásti MPI

MPI-1 – dvouprocesová komunikace (point-to-point communication: send, receive)
– kolektivní komunikace (collective communication: broadcast, scatter, gather, reduce)
– virtuální topologie procesů (kartézská, grafová)
– synchronizace procesů (blokující a neblokující komunikace, bariéry)
– informace o procesech a jejich skupinách (comm, rank, size)

MPI-2 – jednostranná komunikace pro přístup do vzdálené paměti (one-sided communication: put, get, accumulate)
– dynamické řízení procesů pro schéma master-slave (dynamic process management)
– ošetření vstupu a výstupu na distribuovaných systémech (MPI-IO)

Použití MPI ve Fortranu

Pojmy: komunikátor = skupina komunikujících procesů (communicator), identifikace procesu (rank),
„procesor" = proces

Časté argumenty procedur (buf standardního nebo definovaného typu, ostatní integer):

ierror: chybový kód
comm: číslo komunikátoru, komunikátor MPI_COMM_WORLD sdružuje všechny procesy
size: velikost komunikátoru
rank: číslo procesu
buf, sendbuf, recvbuf: proměnná (skalár, pole, struktura) standardního nebo definovaného typu
count: počet prvků v poli buf, 1 pro skalár
datatype: typ proměnné buf, př. konstanty MPI_INTEGER, MPI_REAL, MPI_REAL8, MPI_COMPLEX aj.,
MPI_CHARACTER odpovídá typu CHARACTER(1), víceznakové řetězce jsou ekvivalentní poli znaků,
MPI neprovádí typové konverze (př. INTEGER na REAL), provádí však reprezentační konverze
(př. default REAL na default REAL)
source: číslo zdrojového procesu v komunikaci dvojice, může být MPI_ANY_SOURCE
dest: číslo cílového procesu v komunikaci dvojice
root: číslo zdrojového nebo cílového procesu v kolektivní komunikaci
tag: číslo zprávy, může být MPI_ANY_TAG
status: integer pole o velikosti MPI_STATUS_SIZE, může být MPI_STATUS_IGNORE
request: číslo požadavku při neblokující komunikaci

MPI procedury ve Fortranu jsou implementovány převážně jako podprogramy, existují i funkce a pojmenované konstanty. MPI v C má podobu funkcí vracejících ierror (oproti Fortranu nemají argument ierror).

Inicializace a finalizace, informace

Deklarace MPI jmen zajišťuje vložení souboru mpif.h. Před použitím je třeba knihovnu MPI inicializovat a je vhodné zjistit číslo a velikost komunikátoru a číslo procesu; po použití lze knihovnu finalizovat.

– deklarace	include 'mpif.h' ! může připojovat další soubory: mpi-config.h, mpi-common.h aj.
– inicializace	call MPI_INIT(ierror)
– výchozí komunikátor	comm=MPI_COMM_WORLD
– velikost komunikátoru	call MPI_COMM_SIZE(comm,size,ierror)
– číslo procesu	call MPI_COMM_RANK(comm,rank,ierror)
– finalizace	call MPI_FINALIZE(ierror)

Dvouprocesová komunikace

Odesílatel odesílá jednu zprávu jednomu příjemci ve stejném komunikátoru. Zprávy se skládají z obálky (source, dest, tag, comm) a dat (proměnná buf typu datatype, je-li polem, pak o count prvcích). Přijímaná zpráva nesmí být delší než očekávaná, může být kratší (její aktuální délku vrací MPI_GET_COUNT). Příjemce může přijmout zprávu od konkrétního nebo libovolného odesílatele. Zprávy mezi týmž odesílatelem a příjemcem se nepředbíhají.

Odeslání zprávy (send): MPI_SEND(buf,count,datatype,dest,tag,comm,ierror)
Příjem zprávy (receive): MPI_RECV(buf,count,datatype,source,tag,comm,status,ierror)
Skutečná délka zprávy: MPI_GET_COUNT(status,datatype,count,ierror)

Nezáleží-li příjemci na konkrétním source, tag a status, lze užít předdefinovaných MPI_ANY_SOURCE, MPI_ANY_TAG a MPI_STATUS_IGNORE. Komunikace neproběhne, užije-li se prázdný dest nebo source, tj. MPI_PROC_NULL.

Komunikační módy

Blokující komunikace: příjemce vždy čeká v proceduře pro příjem (receive) až do přijetí celé zprávy, odesílatel vždy čeká v proceduře pro odeslání (send) na odeslání celé zprávy, což není moment totožný s momentem přijetí zprávy příjemcem. Send je obvykle dokončen dříve než párový receive, není však vyloučen opak. Může být úspornější co do času i paměti, je-li receive spuštěn dříve než send.

Módy odesílání:

– buferovaný mód (B): odesílatel zkopíruje zprávu ze své paměti do send-buferu a může pokračovat v běhu nezávisle na připravenosti příjemce, implicitní velikost send-buferu (př. 32 KB) lze explicitně zvětšit, při nedostatečné velikosti send-buferu chyba (buffer overflow),
– synchronní mód (S): odesílatel vyčká na připravenost příjemce (ušetří se bufery a kopírování mezi nimi) a pokračuje až po odeslání zprávy, nebude-li příjemce připraven nikdy, proces uvázne (deadlock),
– ready mód (R): odesílatel předpokládá připravenost příjemce (ušetří se inicializační fáze přenosu), při nepřipraveném příjemci nedefinovaný stav (např. čekání jako v synchronním módu),
– standardní mód: MPI volí podle situace (tj. v závislosti na připravenosti příjemce a dostupnosti buferů), kdy užije buferovaný a kdy synchronní mód.

Příslušné sendy (MPI_BSEND, MPI_SSEND, MPI_RSEND) mají rozhraní totožné s MPI_SEND a párují se s MPI_RECV, který blokuje příjemce až do přijetí celé zprávy. K přenositelnosti programu přispívá, lze-li každý standardní MPI_SEND nahradit synchronním MPI_SSEND, tj. je-li program nezávislý na dostupnosti buferů v daném systému. Při ověřování dostupnosti buferů může být užitečné dočasně nahradit standardní MPI_SEND buferovanými MPI_BSEND. Odesílatel může připojit prostor pro send-bufer explicitně pomocí MPI_BUFFER_ATTACH(buffer,size,ierror) a později odpojit pomocí MPI_BUFFER_DETACH; to nemusí být relevantní pro standardní mód.

Neblokující komunikace (I): send i receive mohou být rozděleny na volání zahajovacího a kompletačního podprogramu (send-start a send-complete, receive-start a receive-complete), mezi kterými může proces vykonávat jinou práci. Mezi voláním send-start a send-complete by odesílatel neměl přepsat odesílanou proměnnou, mezi voláním receive-start a receive-complete by příjemce neměl přijímanou proměnnou ani číst, ani do ní psát. Ukončené odeslání neznamená, že byl dokončen příjem, ukončený příjem neznamená, že bylo dokončeno odeslání. Neblokující volání lze párovat s blokujícími voláními. Neblokující send-start lze kombinovat s módy B, S i R.

Vytvoření požadavku na odeslání nebo příjem:

– odeslání (send-start): MPI_ISEND(buf,count,datatype,dest,tag,comm,request,ierror)
– příjem (receive-start): MPI_IRECV(buf,count,datatype,source,tag,comm,request,ierror)

kde request je integer číslo požadavku (ukazatel na objekt) k užití následnými dotazy a při send-complete,
varianty: MPI_IBSEND, MPI_ISSEND, MPI_IRSEND.

Kompletace požadavku na odeslání nebo příjem:

– odeslání i příjem (send- i receive-complete):

blokující MPI_WAIT(request,status,ierror)
neblokující MPI_TEST(request,flag,status,ierror)
v případě ukončené operace provedou příslušné dealokace a deasociaci requestu a nastaví flag=.true., jinak MPI_WAIT čeká na dokončení operace, zatímco MPI_TEST nastaví flag=.false. a pokračuje,
neblokující MPI_REQUEST_FREE(request,ierror)
provede dealokaci a deasociaci requestu a ponechá možnost (netestovatelného) dokončení operace na pozadí,
varianty pro synchronizaci více požadavků (array_of_requests):
MPI_WAITANY, MPI_TESTANY, MPI_WAITALL, MPI_TESTALL, MPI_WAITSOME, MPI_TESTSOME.

Ostatní podprogramy a funkce:

– zjištění informací o požadavku: MPI_REQUEST_GET_STATUS, MPI_PROBE, MPI_IPROBE,
– zrušení požadavku: MPI_CANCEL.

Perzistentní komunikace pro opakující se přenosy s týmž seznamem argumentů:
MPI_SEND_INIT (s variantami B, S a R) a MPI_RECV_INIT; přenosy zahajuje volání MPI_START(request), kompletuje MPI_WAIT a MPI_TEST (bez dealokace requestu), požadavek dealokuje MPI_REQUEST_FREE.
Kombinace MPI_SENDRECV pro odeslání a příjem dvou zpráv mezi dvěma procesy, případně MPI_SENDRECV_REPLACE pro vzájemnou výměnu jedné zprávy.

Kolektivní komunikace

Určeno pro hromadnou komunikaci uvnitř skupiny nebo mezi skupinami. Vedle synchronizační procedury (BARRIER) sem patří komunikace typu one-to-all (BCAST, SCATTER), all-to-one (GATHER, REDUCE) a all-to-all (ALLTOALL aj.). Nahrazuje sekvence dvouprocesové komunikace.
– synchronizace: MPI_BARRIER(comm)
– rozeslání zprávy: MPI_BCAST(buf,count,datatype, root,comm,ierror)
– rozeslání více zpráv: MPI_SCATTER(sendbuf,sendcount,sendtype, recvbuf,recvcount,recvtype, root,comm,ierror)
– shromáždění zpráv: MPI_GATHER(sendbuf,sendcount,sendtype, recvbuf,recvcount,recvtype, root,comm,ierror)
– redukce: MPI_REDUCE(sendbuf,recvbuf,count,datatype, op, root,comm,ierror)
kde op jsou konstanty MPI_SUM, MPI_PROD, MPI_LAND, MPI_LOR, MPI_MIN, MPI_MINLOC aj.

Definice obecných datových typů

– pole MPI_TYPE_CONTIGUOUS(count,oldtype,newtype)
– struktura MPI_TYPE_CREATE_STRUCT
(count,array_of_blocklengths,array_of_displacements,array_of_types, newtype)
– pro (nepovinné) sbalení nespojitých dat do spojitého buferu
MPI_PACK(inbuf,incount,datatype,outbuf,outsize,position,comm)
MPI_UNPACK(inbuf,insize,position,outbuf,outcount,datatype,comm)

Virtuální topologie procesů

– intrakomunikátory: pro komunikaci uvnitř skupiny procesů
– interkomunikátory: komunikace mezi skupinami

Překladový wrapper mpif90

Volá zvolený fortranský překladač a připojuje potřebné knihovny.
– překlad: mpif90 source.f90
– informace o použitém překladači a nastavení: mpif90 –showme, ompi_info
– nastavení (Open MPI) wrapperu: proměnná OMPI_FC pro překladač, OMPI_FCFLAGS pro volby překladače

Loader mpirun/mpiexec (v Open MPI totéž co orterun)

Spouští požadovaný počet procesů na specifikovaných strojích.
– spuštění 4 procesů na jednom stroji
mpirun -np 4 a.out
– spuštení procesů na více strojích
rozpis slotů na strojích/uzlech v souboru hostfile:
slots pro doporučený počet procesů, max_slots pro maximální počet procesů
počet procesorů počítače: př. cat /proc/cpuinfo
při obsahu souboru hosts: node0 slots=2 max_slots=8
node1 slots=2 max_slots=8
po příkazu mpirun --hostfile hosts -np 8 a.out (volba --byslot je default)
poběží na stroji node0 procesy 0,1,4,5 a na stroji node1 procesy 2,3,6,7 (cyklicky po slotech)
po příkazu mpirun --hostfile hosts -np 8 --bynode a.out
poběží na stroji node0 procesy 0,2,4,6 a na stroji node1 procesy 1,3,5,7 (cyklicky po uzlech)
– běh procesů v agresivním nebo degradovaném režimu
agresivní režim (menší ochota procesu vzdát se procesoru) při počtu procesů ≤ počtu slots
degradovaný režim po překročení počtu slots, tj. při nadobjednávce (oversubscribing) procesů
vynucování agresivního režimu (př. je-li více slots než procesorů) je nevhodné (klesne výkon)
– procesová afinita (mpi_paffinity) pro přesné přiřazení procesů slotům pomocí souboru rankfile (volba –rf rankfile)
př. rankfile: rank 0=node0 slot=0 // rank 1=node1 slot=0-3
pozn. vláknová afinita není, ale vlákna procesu budou omezena na sloty předepsané procesu
pozn. paměťová afinita smysluplná pouze na hardwaru s architekturou NUMA (non-uniform memory access)
– schéma master-slave (multiple programs multiple data, MPMD)
mpif90 -a.out master.f90 ; mpif90 -b.out slave.f90 ; mpirun -np 1 a.out : -np 3 b.out

Ukázky: vzájemná výměna zpráv mezi dvěma procesy (ping-pong)

1. Blokující MPI_RECV
   if (rank==0) then ; call MPI_RECV ; call MPI_SEND ; else ; call MPI_RECV ; call MPI_SEND ; endif
   oba procesy čekají v MPI_RECV na příchod zprávy, která nepřijde: uváznutí (deadlock)
2. Synchronní MPI_SSEND
   if (rank==0) then ; call MPI_SSEND ; call MPI_RECV ; else ; call MPI_SSEND ; call MPI_RECV ; endif
   oba procesy čekají v MPI_SSEND na odezvu příjemce, která nepřijde: deadlock
3. Buferovaný MPI_BSEND
   if (rank==0) then ; call MPI_BSEND ; call MPI_RECV ; else ; call MPI_BSEND ; call MPI_RECV ; endif
   procesy projdou přes MPI_BSEND, pokud mají dostatečně velký bufer, jinak přetečení (buffer overflow)
4. Standardní MPI_SEND
   if (rank==0) then ; call MPI_SEND ; call MPI_RECV ; else ; call MPI_SEND ; call MPI_RECV ; endif
   procesy projdou přes MPI_SEND, pokud mají k dispozici (implicitní!) bufer, jinak deadlock synchronního sendu
5. MPI_SEND a MPI_RECV křížem, kombinovaný MPI_SENDRECV
   a) if (rank==0) then ; call MPI_SEND ; call MPI_RECV ; else ; call MPI_RECV ; call MPI_SEND ; endif
   MPI_RECV čeká a dočká se, MPI_SEND buď buferuje nebo čeká a dočká se: spolehlivé řešení
   b) if (rank==0) then ; call MPI_SENDRECV ; else ; call MPI_SENDRECV ; endif
   MPI_RECV čeká a dočká se, MPI_SEND buď buferuje nebo čeká a dočká se: spolehlivé řešení
6. Ready mód MPI_RSEND
   if (rank==0) then ; call MPI_RSEND ; call MPI_RECV ; else ; call MPI_RECV ; call MPI_RSEND ; endif
   MPI_RSEND předpokládá připravený MPI_RECV, ovšem (v Open MPI) není-li, MPI_RSEND stejně počká
7. Neblokující MPI_ISEND
   a) if (rank==0) then ; call MPI_ISEND ; call MPI_RECV ; call MPI_WAIT ;
   else ; call MPI_ISEND ; call MPI_RECV ; call MPI_WAIT ; endif
   MPI_ISEND nečeká, MPI_RECV se dočká, MPI_WAIT se také dočká: spolehlivé řešení
   b) if (rank==0) then ; call MPI_ISEND ; call MPI_IRECV ; call MPI_WAIT ; call MPI_WAIT ;
   else ; call MPI_ISEND ; call MPI_IRECV ; call MPI_WAIT ; call MPI_WAIT ; endif
   MPI_ISEND nečeká, MPI_IRECV také nečeká, MPI_WAIT se dočkají: spolehlivé řešení

Řešení č. 5 (send a recv křížem nebo kombinovaný sendrecv) je spolehlivé, může však být pomalejší než „duplexní" řešení č. 3 s dostatečně velkým buferem (MPI_ATTACH_BUFFER). Řešení č. 7 je spolehlivé v obou variantách a rychlé alespoň jako č. 3, neboť nemusí nutně buferovat.

6. 5. 2011