Aspekter vedrørende effektiv implementering af MPI

Kort, dansk sammenfatning af

Aspects of the efficient Implementation of the Message Passing Interface (MPI)

indleveret til forsvar for den naturvidenskabelige doktorgrad (Dr. Scient.) ved Københavns Universitet

af

Jesper Larsson Träff

NEC Laboratories Europe, NEC Europe Ltd.
Rathausallee 10, D-53757 Sankt Augustin, Germany
traff@it.neclab.eu

1. November 2007
Antaget til forsvar 29. Januar 2009

MPI ("Message Passing Interface") er den i øjeblikket vigtigste og mest udbredte standard for proceskommunikation på brugerniveau for dedikerede, parallelle systemer med (logisk) fordelt lager. MPI er et bibliotek af brugerfunktioner, der definerer forskellige kommunikationsmodi og de dertil nødvendige abstraktioner for mængder af processer (*kommunikatorer*) og andre fordelte dataobjekter. MPI-standarden opstod i begyndelsen af 90erne og er en konsensus mellem flere sammenlignelige kommunikationsbiblioteker og programmeringssprog for systemer med fordelt lager[1]. Både industri (eksempelvis IBM, Intel, NEC, Meiko) og akademiske institutioner var involverede i standardiseringsprocessen, hvilket har været medvirkende til at gøre standarden både omfattende *og* begrænset til en mindste fællesnævner. MPI udkom første gang i juni 1994, tæt fulgt af en mønsterimplementation, MPICH fra Argonne National Laboratories, USA[2]. Denne første udgave af standarden, MPI-1 omfatter *punkt-til-punkt* (*bilateral*) kommunikation, *kollektiv* kommunikation og *processtopologier*[3]. For alle kommunikationsfunktioner og -modi beskrives strukturen af data enten ved fordefinerede, simple datatyper (heltal, oktetter, flydende-komma tal, etc. svarende til de primitive datatyper i C eller Fortran), eller ved *brugerdefinerede datatyper*, der muliggør (kompakt, rekursiv) beskrivelse af vilkårlig komplekse afbildninger af simple datatyper på lagerudsnit. I 1997 fulgtes MPI-1 af udvidelser for *unilateral* kommunikation, *parallel I/O* og en rudimentær *dynamisk proceshåndtering*. Disse udvidelser betegnes gerne MPI-2[4]. I modsætning til MPI-1, var MPI-2 udvidelserne ikke ledsaget af en mønsterimplementation.

Denne afhandling, som består af ialt 26 videnskabelige artikler, præsenteret på relevante konferencer og i tidsskrifter, beskriver arbejder indenfor næsten alle områder af MPI-standarden (undtaget punkt-til-punkt kommunikation og proceshåndtering), motiveret af problemer med effektiv implementering for NEC's parallelvektorregnere. Denne såkaldte MPI/SX implementation, der benyttes på alle NEC SX-systemer, danner grundlaget for MPI/ES for den såkaldte *Earth Simulator* i Yokohama, Japan, en maskine baseret på SX-arktitekturen med 640 processorknuder med hver 8 vektorprocessorer. *Earth Simulatoren* var ved indvielsen i Februar 2002 den "hurtigste parallelregner i verden" og holdt denne plads i over to år, indtil de første Blue Gene maskiner fra IBM blev operative. MPI/SX er også grundlaget for MPI/EX og MPI/PC for IA64- og 32 bit PC-baserede systemer. Alle disse MPI implementationer er udelukkende til brug for NEC systemer, og koden er ikke tilgængelig for udenforstående.

NEC's SX-processor er en fuld vektorprocessor realiseret på en enkelt CMOS chip (siden SX-6[5]). En vektorprocessor kan udføre aritmetiske og logiske operationer på "vektorer", som kan være tabeller af hel- eller flydende-komma tal, addresseret enten direkte relativt til en startaddresse, eller indirekte via en index-tabel. Hertil kommer specielle, "horizontale" operationer såsom reduktion og søgning. En vektorprocessor både forud-

[1]se: R. Hempel, A. J. G. Hey, O. McBryan, and D. W. Walker. Special issue: Message passing interfaces. *Parallel Computing*, 20(4):415–678, 1994.

[2]se: W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable imlementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996

[3]se: M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI – The Complete Reference*, volume 1, The MPI Core. MIT Press, second edition, 1998.

[4]se: W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir. *MPI – The Complete Reference*, volume 2, The MPI Extensions. MIT Press, 1998.

[5]se: The NEC SX-6 Supercomputer with single-chip vector processor. Kan hentes fra `http://www.hpce.nec.com`

sætter og mulliggør en ekstremt høj lagerbåndbredde (SX-8: 32GBytes/sekund), hvilket resulterer i en ekstremt høj ydeevne (SX-8: 16GFlops[6]), men ikke alle anvendelser lader sig lige nemt, hvis overhovedet, udtrykke med vektoroperationer alene.

SX-arkitekturen er en SMP ("Symmetric MultiProcessor"), bestående af processorknuder med delt lager. Siden SX-6 består en knude af højst 8 processorer. SMP-knuderne er forbundet med et af NEC udviklet fuldt krydsfelt ("single-stage crossbar") kaldet IXS. Alle knuder kan således kommunikere samtidigt, men en knude kan kun være involveret i én kommunikationsoperation med en anden knude ad gangen. IXS netværket er bidirektionalt i den forstand at en knude kan sende data til en knude og samtidig modtage data fra en anden knude, der ikke nødvendigvis behøver at være den samme. Hvis flere processorer fra den samme knude samtidig ønsker at kommunikere, afvikles dette i rækkefølge, sekventielt. IXS netværket kan således karakteriseres som *one-ported, fully connected, full-duplex (send-receive)*. IXS netværket realiserer desuden et mindre antal atomiske registre der kan udnyttes til effektiv synkronisering. En effektiv implementering af MPI for SX-arkitekturen må tage alle disse egenskaber i betragtning.

I det følgende gives en sammenfatning af de i disputatsen indeholdte artikler.

En oversigt over MPI/SX, MPI/ES, og tildels MPI/EX i forskellige udvikingsstadier, særlig hvad angår punkt-til-punkt og kollektiv kommunikation findes i [2, 6]. MPI/SX var formentlig den første fuldstændige implementation af MPI-2 udvidelserne og dermed den fulde MPI standard. Dette er beskrevet i [22], som i særdeleshed omhandler implementeringen af unilateral kommunikation. En portering af unilateral kommunikation (og dermed MPI-2) til et PC system er siden beskrevet i [3]. Allerede fra starten udnyttede MPI/SX de i MPI-2 standarden forudsete optimerings- og tilpasningsmuligheder. Lager, der skal gøres tilgængeligt for andre processer (som del af et *vindue*), kan allokeres "globalt" (med MPI-2 funktionen `MPI_Alloc_mem`) og unilateral kommunikation kan så i mange tilfælde reduceres til kopioperationer. Ligeledes kan den passive synkroniseringsmekanisme under disse omstændigheder implementeres ved atomiske operationer som SX-arkitekturen stiller til rådighed. For lager som ikke er specielt allokeret implementeres unilateral kommunikation ved særlige, interne punkt-til-punkt kommunikationsoperationer. Den kollektive synkroniseringsoperation `MPI_Win_fence` benytter en variant af den kollektive reduktionsoperation `MPI_Reduce_scatter`, som er specielt implementeret til formålet. Alle synkroniseringsfunktioner udnytter de i MPI standarden indførte *assertions*, ofte med hastighedsgevinst.

Behandlingen af brugerdefinerede datatyper i MPI/SX er beskrevet i [19]. De for en MPI implementation væsentlige, interne operationer er pakning og udpakning af data beskrevet ved en brugerdefineret datatype til/fra sammenhængende følger af primitive datatyper. I særdeleshed skal pakning og udpakning kunne ske fra en vilkårlig begyndelsesposition af en struktureret databuffer. Den nye idé i MPI/SX er udskiftning af rekursive kald (som naturligt dikteret af datatypen) med iterativ gennemsøgning af en stak, der kan opbygges med et meget lille antal rekursive kald. Udover at spare et antal rekursive kald, der i andre, trivielle implementationer (som f.eks. i den daværende MPICH implementation), betinget af den måde MPI tillader opbygning af indlejrede typer, kan blive meget stort (ubegrænset), gør stak-implementationen det muligt at vektorisere pakning/udpakning, idet stakken exponerer samtlige regulariteter hvormed de

[6] se: NEC SX-8. Kan hentes fra `http://www.hpce.nec.com`

simple datatyper optræder. For MPI/SX gav denne metode ("flattening on the fly") anledning til køretidsforbedringer af indtil flere størrelsesordner, og brugen af MPI datatyper er i "normale" tilfælde (typisk f.eks. data der kan beskrives som indlejrede vektorer, uanset antallet af niveauer) ikke væsentlig "dyrere" end en manuel ind- og udpakning med til formålet af brugeren skrevne funktioner ville være. Idéen er siden fulgt op af mange arbejder fra andre MPI grupper, men med betydelig forsinkelse.

Til sammenligning af den forbedrede behandling af brugerdefinerede datatyper med andre ansatser udvikledes en samling parametriserbare datatyper af forskellig kompleksitet (indlejrede vektorer, blandede vektorer og strukturer, og lignende). Disse blev indarbejdet i den såkaldte SKaMPI[7] suite [5].

Brugerdefinerede datatyper er centrale i MPI's I/O model. Derfor finder den effektive implementering af ind- og udpakning af data beskrevet ovenfor også anvendelse i implementeringen af MPI-IO, hvilket er beskrevet i [26, 25]. En væsentlig komponent her er flexible funktioner til vilkårlig pakning og udpakning af dele af en datatype, positionering relativt til en datatype og lignende. Dette gør det muligt at slippe af med explicitte lister af primitive data typer (*type maps* i MPI-terminologi), som udover at optage plads ikke tillader effektiv vektorisering.

Ved *kollektiv kommunikation* forstås i MPI-sammenhæng oftest de 16 operationer til synkronisering (`MPI_Barrier`), udveksling af data (`MPI_Bcast`, `MPI_Gather`, `MPI_Alltoall`), og reduktion (`MPI_Reduce`, `MPI_Scan`) beskrevet i standardens kapitel om kollektiv kommunikation. I MPI-sammenhæng betyder "kollektiv" at hvis en proces i en kommunikator kalder en sådan funktion, skal alle andre processer i kommunikatoren kalde den samme funktion *før* eventuelt kald af andre kollektive operationer (på samme kommunikator). I denne forstand er en lang række andre MPI funktioner, f.eks. funktioner til opbygning af nye kommunikatorer, kollektive. For en effektiv implementation af de kollektive synkroniserings-, kommunikations- og reduktionsoperationer antages at alle processer udfører kaldet samtidigt, og i samarbejde udfører den foreskrevne operation.

MPI's dataudvekslingsfunktioner findes alle i *regulære* og *ikke-regulære* udgaver. I de regulære operationer bidrager hver proces med den samme mængde af data (som dog kan være beskrevet ved forskellige brugerdefinerede typer). For de irregulære operationer kan processer bidrage med varierende datamængder, idet dog alle processer parvis skal specificere den samme datamængde.

For effektiv implementering af de kollektive operationer skal der udvikles parallelle algoritmer der er gode relativt til den ovenfor skitserede kommunikationsmodel. I særdeleshed skal SX-systemernes qua SMP hierarkiske kommunikationsstruktur tages i betragtning. At dette sidste er indarbejdet i næsten alle kollektive operationer udmærker stadig MPI/SX i forhold til andre MPI implementationer. De kollektive algoritmer skal skalere (optimalt) med voksende antal af processorer og knuder, have små konstante faktorer, og være praktisk håndtérbare. Som et minimum skal de fungere korrekt under alle mulige omstændigheder.

I det følgende betegner p antallet af MPI processer i den kommunikator over hvilken den kollektive operation udføres, N antallet af SMP knuder, n antallet af MPI processer per knude (ifald dette er det samme over alle knuder), og m den totale problemstørrelse.

I "broadcast" operation `MPI_Bcast` besidder en udvalgt proces data som skal spredes

[7] se: R. Reussner, P. Sanders, and J. L. Träff. SKaMPI: A comprehensive benchmark for public benchmarking of MPI. *Scientific Programming*, 10(1):55–65, 2002.

til de resterende $p-1$ processer. I [14] gives først en algoritme baseret på rekursiv spredning og halvering over et binomial-træ. Paradigmet er kendt, men implementeringen og flere detaljer, f.eks. muligheden for at stoppe halveringen når data bliver "for små" er nye. Algoritmen tager $O(\log p + m)$ skridt. Antages linære kommunikationsomkostninger, altså at overførselstiden for de m data er $t(m) = \alpha + \beta m$, er den linære afhængighed af problemstørrelsen $2\beta m$. Dette er ikke optimalt i fuld duplex kommunikationsmodellen, hvor βm er en nedre grænse.

En optimal "broadcast" algorithme i fuld duplex modellen er opnået i [21, 20], hvor implementationsresultater også gives, der viser en praktisk tidsforbedring forbedring på 50 procent i forhold til f.eks. halveringsalgoritmen ovenfor. I denne algoritme, der indeholder binomialtræ-algoritmen som et specialtilfælde for små problemer, deles de m data i et antal blokke og alle knuder modtager i hver kommunikationsrunde en ny blok og sender en tidligere modtaget blok til en anden knude. Alle knuder er således aktive gennem hele forløbet, og da segmenterne er forholdsvis små kan disse spredes lokalt på knuderne samtidig med/skjult bag kommunikationen mellem knuderne. Implementationsarbejdet viste endvidere at antagelsen af en simpel linær model for kommunikationstiden er helt utilstrækkelig og at langt bedre resultater kan opnås hvis kommunikationsomkostninger istedet modelleres ved stykkevis linære funktioner. Dette har siden vist sig af betydning for tilpasning af andre "pipeline"-baserede algoritmer for "broadcast", reduktion og andre kollektive operationer.

I MPI-sammenhæng har fra andre sammenhænge velkendte sommerfuglevinge ("butterfly") algoritmer i længere tid været anvendt til implementation af reduktionsoperationerne `MPI_Reduce` og `MPI_Alreduce`[8]. Disse algoritmer forudsætter at p er en potens af to, og implementeres oftest med trivielle udvidelser til det generelle tilfælde: en sommerfuglevingealgoritme udføres af et antal processer der udgør den største potens af to der er mindre end eller lig p, og de overskydende processer sender i et første skridt hver især deres data til en proces blandt disse. Dette betyder mindst et ekstra kommunikationsskridt der forøger udførelsestiden proportionalt til m, og de overskydende processer er alle inaktive efter dette første skridt.

Det er altid muligt at reducere denne køretidsforøgelse til at være proportional med $m/2$ (nemlig hvis p er en potens af to plus en extra proces), og generelt er det muligt at holde de overskydende processer beskæftigede med nyttigt arbejde meget længere, hvilket vises i [4], der inholder flere andre forbedringer i stil med "broadcast"-halveringsalgoritmen. Således kan halvering standses når data bliver "for små", og de nye reduktionsalgoritmer indeholder på den måde de velkendte, optimale binomialtræ-algoritmer som specialtilfælde. Implementationsteknisk betyder dette at én og den samme algoritme kan dække hele spektret af problemstørrelser, hvilket overflødiggør vedligeholdelse af forskellige algoritmer ("protokoller") for forskelleige problemstørrelser og andre specialtilfælde.

En sidste variant af reduktionsoperationerne er `MPI_Reduce_scatter`. Dette er en (potentielt) ikke-regulær reduktionsoperation. Det er sommetider blevet hævdet at den ikke i det ikke-kommutative tilfælde kan løses med en sommerfuglevingealgoritme. Denne påstand er forkert, hvilket vises i [15], hvor en simpel proceslokal omordning af input gør det muligt at løse problemet med en sommerfuglevingealgoritme. Er p ikke en potens

[8] se f.eks.: R. van de Geijn. On global combine operations. *Journal of Parallel and Distributed Computing*, 22:324–328, 1994.

af to, kan ideerne ovenfor udnyttes. For at håndtere det ikke-regulære tilfælde, hvor størrelsen af data der sluttelig skal spredes til de enkelte processer varierer, indføres en adaptiv omordning. Antallet af kommunikationsrunder ligger i alle tilfælde mellem $\lceil \log p \rceil$ og $2\lceil \log p \rceil$, jo færre runder jo mere regulært input er.

Forskellige velkendte algoritmer for MPI's *parallelle prefix* operationer `MPI_Scan` og `MPI_Exscan` undersøges i [9]. En ny, simultan "pipeline"-implementation, som udnytter muligheden for bidirektional kommunikation, foreslås og kan for lange vektorer vises at være bedre end kendte "pipeline"-algoritmer. I dette tilfælde er, for små systemer (op til nogle hundrede processorer) en simpel, linær "pipeline" dog væsentlig bedre, grundet små konstantfaktorer. Selv en sådan forefindes såvidt vides ikke i andre MPI implementationer, der ofte baseres på simple eller simultane binomialtræer.

De kollektive operationer for "broadcast", reduktion og parallel prefix kan alle implementeres bekvemt over binærtræer, som også trivielt tillader "pipelining". Binærtræalgoritmer kan imidlertid kun delvist udnytte muligheden for fuld duplex kommunikation, og sådanne algortimer er for både korte og lange data m typisk op til to gange langsommere end teoretisk muligt. Der er to grunde hertil. For det første modtager processer der er blade i det binære træ datablokke, men sender ikke disse videre til andre processer. For det andet sender ikke-blad processer kun en modtaget datablok videre i hver anden kommunikationsrunde, og udnytter således ikke i hver runde muligheden for fuld duplex kommunikation. Dette problem tages op i [7], hvor det vises at begge disse ulemper på naturlig måde forsvinder hvis der anvendes to overlejrede binærtræer istedet for et. De to træer konstrueres på en sådan måde at blade i det ene træ er indre knuder i det andet og omvendt. Det vises at rækkefølgen i hvilken der sendes og modtages data i de to træer kan bestemmes optimalt ved to-farving af en to-regulær, todelt graf. Herved opnås algoritmer for de tre nævnte kollektive operationer der for lange data opnår den teoretisk bedst mulige båndbredde.

For de simple, asymmetriske data samle- og sprede-operationer `MPI_Gather`, `MPI_Scatter` præsenteres i [13] algoritmer baseret på binomial-træer der gradvis bliver mere og mere "flade" med voksende problemstørrelse ("graceful degradation"). Dette giver både optimal kommunikationslatens for små problemer, og optimal udnyttelse af båndbredde for store problemer, samtidig med at den plads der skal afsættes til midlertidige kommunikationsbuffere kan holdes begrænset. Numereres processerne knudevist, kan også sekventialisering af kommunikation mellem knuder holdes begrænset, og de foreslåede algoritmer er derfor direkte anvendelelige for SMP systemer hvor kun en proces ad gangen kan kommunikere med en anden knude. De tilsvarende irregulære operationer `MPI_Gatherv` og `MPI_Scatterv` er implementeret efter samme mønster, men kompliceres af at kun rodprocessen har fuldstændig information. Konstruktionen af det ønskede kommunikationstræ må altså styres fra roden, hvilket betyder en vis omkostning, der dog vises allerede med et beskedent antal knuder at kunne hentes ind igen qua betydelig lavere latens end den linære algoritme der ellers typisk anvendes i MPI biblioteker.

En SMP-egnet algoritme for ikke-individualiseret alle-til-alle (symmetrisk) kommunikation (`MPI_Allgather`) gives i [17]. Ideen her er en hierarkisk dekomposition i lokale sprede-samle operationer, og en global ikke-individualiseret alle-til-alle kommunikation udført af en enkelt proces på hver SMP knude. Som ovenfor kan der kun afsættes begrænset plads til knude-lokale kommunikationsbuffere, og den foreslåede algoritme giver igen en mere flydende overgang mellem to velkendte algoritmer, egnede til henholdsvist

små (logaritmisk latens i antallet af knuder[9]) og store problemer (høj båndbredde).

I [8, 11] behandles dét lastbalanceringsproblem der opstår under regulær, individualiseret alle-til-alle kommunikation (`MPI_Alltoall`) i det tilfælde hvor de enkelte SMP knuder huser et uens antal MPI processer. Det erindres at MPI giver mulighed for vilkårlig dannelse af nye kommunikatorer fra allerede eksisterende via f.eks. `MPI_Comm_split` funktionen, og det kan derfor ske at processerne i en kommunikator er ulige fordelt over de af den givne kommunikator omfattede SMP knuder. En algoritme der anvender paring af processer (faktorisering af kommunikationsgrafen i 1-faktorer) der giver gode resultater for "flade" systemer kan ikke anvendes i dette tilfælde, idet udvekslingen af data mellem to "fede" knuder som implicerer n^2 kommunikationsoperationer vil føre til ventetider for andre knuder. Kommunikation udføres istedet i et antal faser i hvilke der i hver runde udføres nk kommunikationsoperationer hvor k er det mindste antal processer på en knude i den aktuelle fase. Dette betyder at alle fede knuder kan holdes beskæftigede under hele alle-til-alle udvekslingen. Metoden kaldes *hierarkisk faktorisering*.

MPI's kollektive operationer har en ofte kompliceret specifikation og semantik. Da MPI-standarden ikke foreskriver fejlhåndtering i nævneværdig grad, er virkningen af kollektive kald, der ikke er korrekte i henhold til denne, uspecificeret. En anvendelse kan således arbejde videre uden (i første omgang) problemer, gå i baglås ("deadlock", f.eks. ved ikke-konsistent angivelse af rod-proces i reduktions- eller "broadcast"-operation), give forkerte resultater (f.eks. ved angivelse forskellige binære operationer i reduktionsoperation), eller bare afbrydes (f.eks. ved angivelse af parvis forskellige bufferlængder, som kan føre til overskrivning af lagerafsnit). Det er let at se at de foreskrevne betingelser for hver kollektiv operation kan efterprøves ved et antal simplere kollektive kald. F.eks. kan konsistent angivelse af rod-argument efterprøves med en "broadcast"-operation af den i kaldet angivne rod fra en vedtaget proces, som alle processer kan sammenligne med. En sådan omfattende efterprøvelse af kollektive operationer er naturligvis kostbar, især for operationer over små problemstørrelser, og kan derfor ikke tillades under normal programafvikling. Istedet er et særligt verifikationsbibliotek udviklet, der kan benyttes under programudvikling. Dette er beskrevet i [23, 24], og disse ideer blev meget hurtig taget op af Argonne National Laboratories gruppen, der har udviklet lignende biblioteker for `mpich2`.

Af portabilitetsgrunde definerer MPI standarden hverken en køretidsmodel eller giver på anden måde brugeren køretids- eller andre kvalitetsgarantier. Dette valg var fornuftigt og har fremmet udviklingen af en lang række implementationer for højst forskellige systemer og arkitekturer, men kan som ulempe have at brugeren fristes til at udføre MPI-betingede programforbedringer som ikke uden videre vil være fordelagtige for andre MPI implementationer og systemer. Da MPI kommunikationsoperationerne, særlig de kollektive, er semantisk stærkt sammenhængende, og en operation typisk på flere måder kan implementeres ved hjælp af andre, forsøges det i [18] at definere *selv-konsistente køretidskvalitetskrav* som til en vis grad kan overflødiggøre meget specifikke og derfor tvivlsomme programforbedringer. Grundlæggende er ideen at implementationen af enhver i MPI defineret (kollektiv) operation ikke må være værre end en implementation ved hjælp af andre (mere generelle) operationer fra standarden. Disse kvalitetskrav kan

[9] se: J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(11):1143–1156, 1997.

formuleres kvantitativt, og kan således principielt verificeres automatisk for enhver given MPI implementation. Et konkret værktøj er dog ikke udviklet.

MPI suggererer brugeren et "fladt" system hvori alle processer kan kommunikere med alle andre processer med de samme omkostninger. For systemer med et ikke-homogent kommunikationsnetværk eller for SMP systemer, der har en hierarkisk kommunikationsstruktur, er dette ikke tilfældet. MPI standarden indeholder en simpel mekanisme der har til hensigt at muliggøre portabel tilpasning til ikke-homogene systemer. Denne mekanisme består i brugerdefinerede procestopologier. Brugeren kan beskrive sin anvendelses kommunikationsmønster som en ikke-orienteret, ikke-vægtet kommunikationsgraf. MPI kan med denne generere en procesomordning i form af en ny kommunikator, hvori kommunikerende processer ligger "tæt" ved hinanden i det underliggende fysiske kommunikationssystem (f.eks. på samme SMP knude). I [10] vises det at omordningsproblemet for SMP systemer kan formuleres som et grafdelingsproblem i N delmængder af størrelse (højst) n. Denne formulering er indbygget i MPI/SX, der er en af få (hvis overhovedet andre) MPI implementationer med en ikke-triviel implementation af procestopologifunktionaliteten.

En forbedret heuristik for grafdeling i k delmængder (svarende til antallet og størrelsen af de SMP knuder som dækkes af den i kaldet angivne kommunikator) er udviklet og beskrevet i [16]. Denne generalisering af Fiduccia-Mattheyses varianten af Kernighan-Lin heuristikken for grafbipartitionering til k-partitionering med den samme komplexitet (rekursiv bibipartitionering kan resultere i vilkårlig forværring, og tager i værste fald en logaritmisk faktor længere tid[10]) er af selvstændig interesse.

MPI-standarden for procestopologier gør en række forsimplede antagelser som i høj grad indskrænker den mulige nytte af denne funktionalitet. Specifikationen er f.eks. ikke skalérbar, og det er ikke muligt at skelne mellem "tunge" og "lette" kommunikationsforbindelser. Denne kritik kan findes i [12], og senere, og mere udbygget i [1]. I sidstnævnte vises det ved simple modeksempler at gode partitioneringer for anvendelser med hierarkisk kommunikationsstruktur ikke kan findes uden som minimum at tillade vægtede kanter. Desuden foreslås en ortogonal løsning hvor MPI istedet leverer information som kan benyttes af en selvstændig partitioneringspakke.

MPI/SX er et produktionsbibliotek, som til stadighed er under udvikling og forbedring. Udover ydeevne og god udnyttelse af SX-systemet er korrekthed og robusthed af eminent vigtighed. De her udvalgte artikler repræsenter derfor kun en side af arbejdet med MPI, og kun et øjebliksbillede af NEC's MPI implementationer. Tidlige algoritmer og implementationer er forbedret og forfinet, hvilket ikke altid kan aflæses af de publicerede resultater, og dele er naturligvis ikke egnede for en bredere offentlighed. Sammenlagt formidler de her samlede arbejder dog et indtryk af det niveau hvorpå MPI/SX og NEC's øvrige MPI biblioteker befinder sig, også i forhold til andre, åbent tilgængelige MPI biblioteker. Aktuelt arbejdes på forbedringer af den irregulære, ikke-personaliserede alle-til-alle kommunikationsoperation baseret på de opnåede "broadcast"-resultater. Andre algoritmiske problemer vedrører kommunikationsoptimal reduktion og parallel præfix, samt irregulær, personaliseret alle-til-alle kommunikations, hvor ingen praktisk gode algoritmer synes at findes. Af stor vigtighed er tilpasning til nye kommunikationsnetværk, navnlig for MPI/EX, men som også kan forudses for kommende SX-systemer.

MPI-standarden selv, som nu har eksisteret i over 10 år, vil, trods kritik fra mange

[10] se: H. Simon and S.-H. Teng. How good is recursive bisection. *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.

sider, spille en uomgængelig rolle i de kommende år, både for det daglige arbejde med kendte anvendelser og systemer og for nye, planlagte "Petaflop" systemer, både i USA og Japan, men forhåbentlig også i Europa. En grund hertil er at standarden selv og de bedste implementationer heraf har vist sig forbløffende skalérbare; dette har været en vigtig erfaring fra Earth Simulator projektet og senest IBM's Blue Gene. Mindre korrekturer står forude, men det er uklart om der er behov eller interesse for større tilføjelser, eller om de problemer der ikke kan håndteres med MPI bedre løses med andre tiltag. Mulige udvidelser af MPI er tænkelige på følgende punkter: model for robusthed ("fault tolerance"), funktionalitet for bedre tilpasning til stærkt heterogene systemer, og bedre, mere systematisk understøttelse af asynkronicitet og latensoverdækning.

Disputatsen består af en mere omfattende oversigt på Engelsk fulgt af de ovenfor kort resúmerede artikler i den form hvori de har været bragt.

Litteratur

[1] Guntram Berti and Jesper Larsson Träff. What MPI could (and cannot) do for mesh-partitioning on non-homogeneous networks. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 13th European PVM/MPI Users' Group Meeting*, volume 4192 of *Lecture Notes in Computer Science*, pages 293–302. Springer, 2006.

[2] Maciej Gołebiewski, Hubert Ritzdorf, Jesper Larsson Träff, and Falk Zimmermann. The MPI/SX implementation of MPI for NEC's SX-6 and other NEC platforms. *NEC Research & Development*, 44(1):69–74, 2003.

[3] Maciej Gołebiewski and Jesper Larsson Träff. MPI-2 one-sided communications on a Giganet SMP cluster. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 8th European PVM/MPI Users' Group Meeting*, volume 2131 of *Lecture Notes in Computer Science*, pages 16–23. Springer, 2001.

[4] Rolf Rabenseifner and Jesper Larsson Träff. More efficient reduction algorithms for message-passing parallel systems. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 11th European PVM/MPI Users' Group Meeting*, volume 3241 of *Lecture Notes in Computer Science*, pages 36–46. Springer, 2004.

[5] Ralf Reussner, Jesper Larsson Träff, and Gunnar Hunzelmann. A benchmark for MPI derived datatypes. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 7th European PVM/MPI Users' Group Meeting*, volume 1908 of *Lecture Notes in Computer Science*, pages 10–17. Springer, 2000.

[6] Hubert Ritzdorf and Jesper Larsson Träff. Collective operations in NEC's high-performance MPI libraries. In *International Parallel and Distributed Processing Symposium (IPDPS 2006)*, page 100, 2006.

[7] Peter Sanders, Jochen Speck, and Jesper Larsson Träff. Full bandwidth broadcast, reduction and scan with only two trees. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 14th European PVM/MPI Users' Group Meeting*, volume 4757 of *Lecture Notes in Computer Science*, pages 17–26. Springer, 2007. (**Outstanding paper**).

[8] Peter Sanders and Jesper Larsson Träff. The hierarchical factor algorithm for all-to-all communication. In *Euro-Par 2002 Parallel Processing*, volume 2400 of *Lecture Notes in Computer Science*, pages 799–803. Springer, 2002.

[9] Peter Sanders and Jesper Larsson Träff. Parallel prefix (scan) algorithms for MPI. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 13th European PVM/MPI Users' Group Meeting*, volume 4192 of *Lecture Notes in Computer Science*, pages 49–57. Springer, 2006.

[10] Jesper Larsson Träff. Implementing the MPI process topology mechanism. In *Supercomputing*, 2002. `http://www.sc-2002.org/paperpdfs/pap.pap122.pdf`.

[11] Jesper Larsson Träff. Improved MPI all-to-all communication on a Giganet SMP cluster. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 9th European PVM/MPI Users' Group Meeting*, volume 2474 of *Lecture Notes in Computer Science*, pages 392–400. Springer, 2002.

[12] Jesper Larsson Träff. SMP-aware message passing programming. In *Eigth International Workshop on High-level Parallel Programming Models and Supportive Environments (HIPS03), International Parallel and Distributed Processing Symposium (IPDPS 2003)*, pages 56–65, 2003.

[13] Jesper Larsson Träff. Hierarchical gather/scatter algorithms with graceful degradation. In *International Parallel and Distributed Processing Symposium (IPDPS 2004)*, page 80, 2004.

[14] Jesper Larsson Träff. A simple work-optimal broadcast algorithm for message-passing parallel systems. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 11th European PVM/MPI Users' Group Meeting*, volume 3241 of *Lecture Notes in Computer Science*, pages 173–180. Springer, 2004.

[15] Jesper Larsson Träff. An improved algorithm for (non-commutative) reduce-scatter with an application. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 12th European PVM/MPI Users' Group Meeting*, volume 3666 of *Lecture Notes in Computer Science*, pages 129–137. Springer, 2005.

[16] Jesper Larsson Träff. Direct graph k-partitioning with a Kernighan-Lin like heuristic. *Operations Research Letters*, 34(6):621–629, 2006.

[17] Jesper Larsson Träff. Efficient allgather for regular SMP-clusters. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 13th European PVM/MPI Users' Group Meeting*, volume 4192 of *Lecture Notes in Computer Science*, pages 58–65. Springer, 2006.

[18] Jesper Larsson Träff, William Gropp, and Rajeev Thakur. Self-consistent MPI performance requirements. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 14th European PVM/MPI Users' Group Meeting*, volume 4757 of *Lecture Notes in Computer Science*, pages 36–45. Springer, 2007. (**Outstanding paper**).

[19] Jesper Larsson Träff, Rolf Hempel, Hubert Ritzdorf, and Falk Zimmermann. Flattening on the fly: efficient handling of MPI derived datatypes. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 6th European PVM/MPI*

Users' Group Meeting, volume 1697 of *Lecture Notes in Computer Science*, pages 109–116. Springer, 1999.

[20] Jesper Larsson Träff and Andreas Ripke. An optimal broadcast algorithm adapted to SMP-clusters. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 12th European PVM/MPI Users' Group Meeting*, volume 3666 of *Lecture Notes in Computer Science*, pages 48–56. Springer, 2005.

[21] Jesper Larsson Träff and Andreas Ripke. Optimal broadcast for fully connected networks. In *High Performance Computing and Communications (HPCC'05)*, volume 3726 of *Lecture Notes in Computer Science*, pages 45–56. Springer, 2005.

[22] Jesper Larsson Träff, Hubert Ritzdorf, and Rolf Hempel. The implementation of MPI-2 one-sided communication for the NEC SX-5. In *Supercomputing*, 2000. `http://www.sc2000.org/proceedings/techpapr/index.htm#01`.

[23] Jesper Larsson Träff and Joachim Worringen. Verifying collective MPI calls. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 11th European PVM/MPI Users' Group Meeting*, volume 3241 of *Lecture Notes in Computer Science*, pages 18–27. Springer, 2004.

[24] Jesper Larsson Träff and Joachim Worringen. The MPI/SX collectives verification library. In *Parallel Computing: Current & Future Issues of High-End Computing (ParCo 2005)*, volume 33 of *NIC Series*, pages 909–916. John von Neuman Institute for Computing, Central Institute for Applied Mathematics, Forschungszentrum Jülich, 2006.

[25] Joachim Worringen, Jesper Larsson Träff, and Hubert Ritzdorf. Fast parallel non-contiguous file access. In *Supercomputing*, 2003. `http://www.sc-conference.org/sc2003/tech_papers.php`.

[26] Joachim Worringen, Jesper Larsson Träff, and Hubert Ritzdorf. Improving generic non-contiguous file access for MPI-IO. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 10th European PVM/MPI Users' Group Meeting*, volume 2840 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2003.

Bidragene [7, 18, 21, 20] er siden indleveringen af denne disputats udkommet eller akcepteret til udgivelse i tidsskrifter, som beskrevet nedenfor. Disse udgaver kan med fordel læses fremfor de citerede konferencebidrag.

- Jesper Larsson Träff and Andreas Ripke. Optimal broadcast for fully connected processor-node networks. *Journal of Parallel and Distributed Computing*, 68(7):887–901, 2008. Læses istedet for [21, 20].
- Peter Sanders, Jochen Speck, and Jesper Larsson Träff. Two-Tree Algorithms for Full Bandwidth Broadcast, Reduction and Scan. *Parallel Computing*, to appear 2009. Læses istedet for [7].

- Jesper Larsson Träff, William D. Gropp, Rajeev Thakur. Self-consistent MPI Performance Guidelines. *IEEE Transactions on Parallel and Distributed Systems*, to appear 2009. Læses istedet for [18].