ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1997
Seria: INFORMATYKA z. 33 Nr kol. 1377

Mateusz NOWAK
Instytut Informatyki Teoretycznej i Stosowanej PAN

PORÓWNANIE CECH ŚRODOWISK PVM I MPI

Streszczenie. Artykuł przedstawia dwa obecnie najpopularniejsze standardy środowisk do tworzenia programów równoległych opartych na modelu z przesyłem wiadomości - MPI (Message Passing Interface) i PVM (Parallel Virtual Machine). Oba środowiska porównano, wskazując te ich cechy, które w różnych sytuacjach mogą być uważane za zalety i wady.

COMPARISON OF FEATURES OF PVM AND MPI ENVIRONMENTS

Summary. Article presents the two most popular standards of environments for parallel programming with use of message passing paradigm: MPI (Message Passing Interface) and PVM (Parallel Virtual Machine). Both environments are compared, with pointing at their advantages and disadvantages in various situations.

1. Wstęp

W chwili obecnej w rozwoju techniki przetwarzania danych znajdujemy się w punkcie, w którym przetwarzanie rozproszone coraz bardziej zyskuje na znaczeniu. Zarówno wielkość zadań obliczeniowych, jak i ilość gromadzonych danych coraz częściej przekracza możliwości pojedynczych procesorów i pojedynczych komputerów. Również w przypadku, gdy istnieją komputery o odpowiedniej mocy, zastosowanie wielu prostych węzłów obliczeniowych jest po prostu tańsze od wykorzystania dużego komputera o analogicznych możliwościach obliczeniowych. Z tych powodów coraz częściej obecnie sięga się do systemów rozproszonych. Zjawisko to obserwujemy w obliczeniach największej skali, do których zaprzęga się ogólnokrajową sieć superkomputerów. Działającym już przykładem może być

stworzone w USA tzw. MetaCentrum ([3]), obejmujące swoim zasięgiem największe centra superkomputerowe kraju połączone szybką siecią ATM umożliwiającą efektywną komunikację. Dzięki stworzeniu takiej struktury można rozwiązywać problemy obliczeniowe, których rozmiary dotychczas przekraczały możliwości któregokolwiek pojedynczego komputera. Również w budowie komputerów o dużej mocy obliczeniowej odchodzi się od projektowania procesorów o dużej wydajności jednostkowej (np. procesory wektorowe) na rzecz komputerów zbudowanych z wielu stosunkowo tanich procesorów (tzw. komputery masywnie równoległe, np. IBM SP2 [13], Cray T3D [9] czy Meiko CS-2HA [1]).

Podobne działania prowadzi się też w zastosowaniach o mniejszej skali, w których taniej jest zastosować sieć stacji roboczych w miejsce komputera dużej czy średniej mocy. Grona stacji roboczych stosuje się już szeroko do obliczeń naukowych (np. projekt *NOW - Network of Workstations*,[14]), pojawiają się one też już w rozwiązaniach komercyjnych (np. wyszukiwarka internetowa HotBot firmy Inktomi Corporation [2]).

Z wymienionych wyżej powodów coraz pilniejsza staje się potrzeba stworzenia i spopularyzowania oprogramowania systemów wieloprocesorowych, umożliwiającego stosunkowo łatwe tworzenie aplikacji działających efektywnie w takich właśnie systemach.

2. Środowiska do przetwarzania rozproszonego

Architektura budowanych obecnie systemów wieloprocesorowych zazwyczaj odpowiada modelowi komputera wieloprocesorowego z rozdzieloną pamięcią ([18]). W takim modelu komputer składa się z wielu procesorów, z których każdy ma własną pamięć operacyjną. Procesory te połączone są podsystemem komunikacyjnym, umożliwiającym przesył danych pomiędzy nimi. Podstawową cechą takiej architektury, która jak się wydaje, przesądziła o jej sukcesie, jest jej łatwa skalowalność, a więc możliwość zwiększania mocy obliczeniowej przez proste dodawanie kolejnych modułów. W ogólnym przypadku w komputerze o architekturze z rozdzieloną pamięcią jest to możliwe w znacznie większej skali i przy niższym koszcie, niż w systemach z pamięcią wspólną. Prowadzi to do sytuacji, w której większość produkowanych obecnie dużych systemów wieloprocesorowych albo wprost odpowiada modelowi z rozdzieloną pamięcią (komputery masywnie równoległe), albo ma budowę hybrydową, gdzie poszczególne węzły obliczeniowe zbudowane są z kilku (zazwyczaj 2 do 4) procesorów wyposażonych we wspólną pamięć, jednak poszczególne węzły między sobą mogą komunikować się wyłącznie przez podsystem komunikacyjny, tak jak w modelu z pamięcią rozdzieloną (np. Sun Enterprise 10 000 [12], Cray T3E [10], SGI Origin 2000 [8]). Taki hybrydowy model umożliwia budowanie komputerów najwyższej mocy. Największe istniejące konfiguracje komputerów równoległych mają kilka tysięcy procesorów ([11]).

Klasą maszyn, która osiąga obecnie coraz większe znaczenie, a którą też trzeba zaliczyć do maszyn z rozdzieloną pamięcią, są grona (ang. *clusters*) stacji roboczych. Okazuje się bowiem, że komputery klasy stacji roboczych, produkowane masowo, oferują najniższy koszt obliczeń liczony w dolarach za 1 Mflop dostępnej mocy obliczeniowej (por. [14]). Łączenie takich stacji za pomocą wydajnej sieci komputerowej pozwala uzyskiwać moce obliczeniowe odpowiadające średnim i dużym komputerom.

Podstawowym modelem programowania systemów z rozdzieloną pamięcią jest model z przesyłem wiadomości lub komunikatów (ang. *message passing*). Model ten jest najbliższy rzeczywistej architekturze i sposobowi działania komputera i dzięki temu programy rozproszone, pisane z wykorzystaniem przesyłu wiadomości, mogą być efektywne. Model ten jest niestety mniej przyjazny dla programisty niż model ze wspólną pamięcią. Jednak środowiska umożliwiające programowanie z wykorzystaniem modelu wspólnej pamięci muszą działać na pamięci fizycznie rozproszonej. Wspólna pamięć w rzeczywistości jest emulowana za pomocą przesyłu wiadomości, co w oczywisty sposób wpływa na obniżenie efektywności. W chwili obecnej środowiska pozwalające programiście na wykorzystanie modeli programowania bardziej przyjaznych niż przesył wiadomości, a pracujące na maszynach z rozdzieloną pamięcią znajdują się jeszcze we wczesnym stadium rozwoju (por. [19]). Dopóki ta sytuacja nie ulegnie zmianie, w programowaniu maszyn równoległych, zarówno klasy superkomputerowej, jak i gron stacji roboczych, bardzo duże znaczenie będą miały właśnie środowiska udostępniające programiście model maszyny równoległej z przesyłem wiadomości.

3. Standardy PVM i MPI

Systemy operacyjne, w które wyposażane są obecnie komputery, oferują niewielkie możliwości w zakresie komunikacji między komputerami lub oferowane możliwości są specyficzne dla danej platformy sprzętowo-programowej. Z drugiej strony wiele programów pisanych jest w językach uniwersalnych, dla których kompilatory istnieją w niemal wszystkich środowiskach, przede wszystkim w językach Fortran i C. Sytuacja taka powodowała, że można było pisać programy przeznaczone do wykonania w różnych systemach komputerowych, jednak programy te nie mogły zawierać prostych w użyciu, przenośnych mechanizmów przesyłu komunikatów między węzłami obliczeniowymi. Naprzeciw tym wymaganiom wyszły standardowe, od strony programisty jednakowe we wszystkich systemach, biblioteki uzupełniające możliwości oferowane przez języki programowania o funkcje przesyłania wiadomości.

Najpopularniejszą obecnie specyfikacją biblioteki funkcji uzupełniających funkcjonalność systemów operacyjnych o możliwości przesyłu wiadomości jest biblioteka PVM ([16], ang. *Parallel Virtual Machine* - równoległa maszyna wirtualna). Ilość jej implementacji

w różnych środowiskach operacyjnych pozwala mówić o specyfikacji PVM jako o standardzie *de facto*.

Dużą popularność osiąga w ostatnim okresie również inna specyfikacja biblioteki o podobnej funkcjonalności - MPI ([5], ang. *Message Passing Interface* - interfejs przesyłu komunikatów). MPI został zaprojektowany jako propozycja standardu *de jure* dla producentów oprogramowania implementujących model z przesyłem komunikatów. Każda z tych dwóch bibliotek powstała w inny sposób, inny jest też ich repertuar usług, cechy użytkowe i przeznaczenie. Istnieją również inne środowiska umożliwiające programowanie zgodnie z modelem przesyłu komunikatów, jednak ich zastosowanie jest ograniczone do wybranych, specyficznych środowisk i z tego względu nie mają one szans na osiągnięcie większej popularności.

Idea PVM powstała w 1989 r. w Oak Ridge National Laboratory. Kolejne wersje powstawały w 1991 r. (PVM 2.0) i w 1993 r. (PVM 3.0). Interfejs programisty ewoluował więc stopniowo tak, by zaspokoić oczekiwania poszerzającego się grona użytkowników i nadążać za rozwijającą się w szybkim tempie technologią sieciową i obliczeniową. Początkowo PVM przeznaczony był dla gron stacji roboczych, później zaczął być implementowany również na dużych komputerach wieloprocesorowych.

Inna była historia zdefiniowania interfejsu programisty MPI. Został on stworzony przez komitet nazwany MPI Forum. W jego skład weszli eksperci w obliczeniach równoległych z około 40 ośrodków zajmujących się tą problematyką, zarówno naukowych, jak i komercyjnych. Nacisk na stworzenie standardu powstał, gdy każdy z producentów systemów masywnie równoległych tworzył własne środowisko do tworzenia programowania wykorzystującego przesył komunikatów. MPI miał być standardową specyfikacją, którą implementowałby każdy z producentów systemów MPP. Celem twórców MPI było zaprojektowanie takiej specyfikacji, której implementacje mogłyby korzystać z najszybszych dostępnych w danym środowisku mechanizmów, dlatego zdecydowano się poświęcić pewne cechy funkcjonalne (np. możliwość współpracy pomiędzy różnymi implementacjami) na korzyść wyższej szybkości działania. MPI został zaimplementowany również dla sieci stacji roboczych.

Użytkownicy implementacji standardu MPI wkrótce wskazali na braki w jego specyfikacji. W 1995 roku MPI Forum rozpoczęło spotkania w celu zaprojektowania uzupełnionej specyfikacji MPI 2. W chwili obecnej oczekuje się na opublikowanie pełnego standardu MPI 2, a pierwszą wersję MPI często określa się jako MPI 1.

W kolejnych rozdziałach przedstawiono dokładniej biblioteki MPI i PVM z uwzględnieniem różnic między nimi.

4. Porównanie cech użytkowych

Zarówno PVM, jak i MPI uzupełniają możliwości systemów operacyjnych, przede wszystkim klasy UNIX, o możliwości komunikacji między procesami działającymi na różnych węzłach komputera równoległego (sieci komputerów). Cechy i możliwości udostępniane przez te biblioteki użytkownikowi różnią się jednak w znacznym stopniu. W bieżącym rozdziale przedstawiono i porównano możliwości obu bibliotek nie tylko w zakresie przesyłu wiadomości pomiędzy procesami, ale też współpracy między różnymi systemami komputerowymi oraz organizacji przetwarzania równoległego.

4.1. Heterogeniczność a systemy PVM i MPI

W obliczeniach dużej mocy możliwość pracy w środowiskach heterogenicznych staje się coraz bardziej istotna. Wiele instytucji używa wielu różnych systemów komputerowych, zakupionych w różnym okresie i dla różnych celów. Systemy z przesyłem komunikatów, umożliwiające programowanie grup komputerów połączonych przez sieć komputerową, powinny umożliwiać wspólne wykorzystanie takich komputerów i połączenie ich mocy obliczeniowej. Istotna jest również przenośność oprogramowania, kiedy np. chcemy udostępnić oprogramowanie użytkownikom innych systemów czy po prostu przenieść naszą aplikację w środowisko o większej mocy obliczeniowej.

Interfejs MPI został zaprojektowany tak, aby mógł zostać zaimplementowany w różnych środowiskach MPP i gron stacji roboczych. Przenośność oferowana przez MPI oznacza, że program napisany dla jednej architektury może być w innym systemie bez przeszkód skompilowany i wykonany - bez modyfikacji kodu źródłowego.

Również PVM zapewnia ten poziom przenośności. W przypadku biblioteki PVM do przenośności dochodzi jeszcze możliwość współpracy pomiędzy różnymi jej implementacjami. Programy napisane z wykorzystaniem PVM, podobnie jak w przypadku MPI, mogą zostać przeniesione i skompilowane bez modyfikacji w innym systemie. Dodatkowo jednak powstałe w ten sposób programy z różnych systemów mogą komunikować się za sobą. Innymi słowy, aplikacja wykorzystująca MPI może być uruchomiona, jako całość, w dowolnej z wielu architektur równoległych i w tym sensie jest przenośna. Programy korzystające z PVM natomiast mogą współpracować pomiędzy różnymi architekturami. Standard MPI, co prawda, nie zabrania stworzenia specyfikacji umożliwiającej taką współpracę, jednak jej nie wymaga, zaś implementacja uniwersalnego interfejsu komunikacyjnego spowodowałaby spowolnienie działania całego systemu. Jedynie te implementacje MPI dla stacji roboczych, które zostały zaprojektowane z myślą o pracy w różnych środowiskach (np. systemy Unix na stacjach różnych producentów), umożliwiają wzajemną komunikację pomiędzy różnymi stacjami robo-

czymi, pod warunkiem, że na każdej z nich jest zainstalowana ta sama implementacja MPI. Możliwe jest dzięki temu tworzenie heterogenicznych gron stacji roboczych.

Brak definicji odpowiedniego protokołu komunikacyjnego wspólnego dla wszystkich implementacji MPI bierze się z założonego przy projektowaniu priorytetu dla wydajności i szybkości działania. Najlepszym sposobem, by system przesyłu wiadomości był w danej architekturze najszybszy, jest wykorzystanie jej własnych, specyficznych cech sprzętowych i rezygnacja z jakichkolwiek niepotrzebnych w danym środowisku narzutów. Takie założenie prowadzi jednak do niekompatybilności na poziomie komunikacji pomiędzy różnymi systemami komputerowymi.

Podczas projektowania PVM zdecydowano się na poświęcenie części możliwej do osiągnięcia wydajności na rzecz możliwości komunikowania się pomiędzy systemami o różnych architekturach. PVM rozpoznaje przeznaczenie wiadomości i na tej podstawie określa sposób, w jaki należy ją przesłać. Jeżeli komunikat przesyłany jest w obrębie danego systemu lub do systemu o identycznej architekturze, do jego transportu wykorzystywane są własne mechanizmy komunikacyjne. Jeżeli natomiast użycie tych mechanizmów nie jest możliwe, gdyż wiadomość przeznaczona jest dla systemu innego typu, używany jest wspólny dla wszystkich implementacji PVM protokół.

PVM i MPI różnią się też w podejściu do współpracy pomiędzy programami napisanymi w różnych językach. Oba te systemy zapewniają interfejs programowy dla języków C i Fortran. W PVM program napisany w C może wysyłać i odbierać komunikaty od programu napisanego w Fortranie, standard MPI nie stawia natomiast takiego wymagania.

4.2. Maszyna wirtualna

Maszyna wirtualna to logiczny zbiór zasobów komputerowych widzianych i zarządzanych przez programistę jak pojedynczy, równoległy komputer. Zasoby te fizycznie mogą być częściami jednego komputera bądź mogą to być różne komputery połączone za pośrednictwem sieci. Pojęcie maszyny wirtualnej jest postawą filozofii PVM. W przypadku tej biblioteki maszyna wirtualna jest dynamiczna i heterogeniczna, tzn. poszczególne jej części (węzły obliczeniowe) mogą być dynamicznie włączane i usuwane z maszyny podczas wykonywania zadania i mogą być systemami o różnych cechach i architekturach. Koncepcja taka pozwala łączyć w jeden, zintegrowany zasób obliczeniowy stacje robocze różnych typów, komputery osobiste i komputery masywnie równoległe.

W przeciwieństwie do PVM, MPI skupia się na przesyłaniu wiadomości. Twórcy MPI wprost stwierdzili, że zarządzanie zasobami i koncepcja maszyny wirtualnej znajdują się poza zainteresowaniem standardu MPI, tak wersji 1, jak i 2. Z tego względu możliwości oferowane przez dynamiczną maszynę wirtualną PVM są znacznie szersze, niż możliwości MPI w tym

zakresie. Zasoby obliczeniowe, czyli komputery biorące udział w procesie obliczeniowym, mogą być dodawane i usuwane z maszyny wirtualnej w dowolnym momencie. Jest to możliwe zarówno za pośrednictwem konsoli systemu, jak i z wnętrza aplikacji. Takie mechanizmy dają użytkownikowi silne narzędzie dla projektowania aplikacji z uwzględnieniem problemów wyrównywania obciążenia, migracji zadań i odporności na uszkodzenia. Dzięki usługom nazw (ang. *name services*) dostarczanym przez maszynę wirtualną PVM niezależnie uruchamiane zadania mogą odnaleźć się nawzajem i komunikować się.

Istotną częścią pracy maszyny wirtualnej jest również sterowanie procesami i ich kontrola. PVM zawiera bogaty repertuar funkcji zarządzania procesami, takie jak możliwości uruchamiania i zatrzymywania zadań czy sprawdzanie, które procesy i gdzie aktualnie działają. MPI 1 nie zawiera żadnych tego typu mechanizmów, MPI 2 natomiast będzie zawierał funkcje do uruchamiania i zatrzymywania (przerywania) grup zadań, być może również inne.

Inny aspekt dynamicznej maszyny wirtualnej związany jest z efektywnością. Aplikacje użytkownika mogą mieć zmieniające się w trakcie wykonania potrzeby obliczeniowe. Mechanizmy dostarczane przez PVM umożliwiają elastyczne wykorzystanie dostępnych zasobów obliczeniowych. Przykładowa aplikacja, która rozpoczyna się i kończy obliczeniami sekwencyjnymi, ale zawiera fragment, który może być mocno zrównoleglony, nie musi rezerwować już na początku swojego wykonania dużej maszyny masywnie równoległej. Komputer równoległy może zostać włączony do maszyny wirtualnej w momencie, gdy zaczyna być potrzebny, a po wykorzystaniu może być z niej usunięty. Poza tym czasem komputer ten może być wykorzystywany do innych celów. Podobną sytuację możemy sobie wyobrazić, gdy uruchamiana jest aplikacja wykonująca długotrwałe obliczenia, a użytkownik chce tylko od czasu do czasu mieć wgląd w wyniki pośrednie. Stacja graficzna wizualizująca wyniki nie musi więc przez cały czas być zarezerwowana na potrzeby aplikacji - wystarczy na żądanie użytkownika włączyć ją do zbioru komputerów tworzących maszynę wirtualną wykonującą obliczenia, dzięki czemu możliwe stanie się pobranie wyników pośrednich i pokazanie ich w wybrany przez użytkownika sposób.

Koncepcja maszyny wirtualnej ułatwia również użytkownikowi zarządzanie zasobami obliczeniowymi. Użytkownik nie musi oddzielnie rejestrować się na każdej z maszyn stanowiących maszynę wirtualną, uruchamiać właściwych dla tej maszyny procesów i kontrolować ich wykonania. Maszyna wirtualna stanowi pewną prostą abstrakcję obejmującą różne komputery i zasoby, podzieloną na warstwy umożliwiające kontrolę i sterowanie na różnych poziomach. Użytkownik może utworzyć zbiór maszyn i korzystać z nich jak z jednakowych węzłów obliczeniowych niezależnie od różnic w ich rzeczywistej architekturze, może też zwiększyć poziom szczegółowości i zażądać, aby pewne zadania wykonywać na komputerach o określonych cechach lub nawet na konkretnej maszynie.

Standard MPI nie przewiduje żadnego poziomu abstrakcji dla zasobów obliczeniowych i pozostawia konkretnej implementacji lub użytkownikowi stworzenie własnego systemu zarządzania. Taki stworzony na własne potrzeby system może być ostatecznie lepszy i bardziej wygodny dla konkretnych potrzeb użytkownika, ale jego stworzenie wymaga znacznego nakładu pracy.

Jedyną płaszczyzną standardu MPI, na której mamy do czynienia z pewną abstrakcją ukrywającą fizyczną warstwę systemu, jest topologia przesyłu wiadomości. Grupa zadań MPI może być zorganizowana w określoną topologię, zgodnie z którą później przesyłane są wiadomości. Jeżeli topologia ta będzie zgodna z rzeczywistą topologią fizycznej sieci połączeń, przesył wiadomości będzie najefektywniejszy.

4.3. Przesył komunikatów

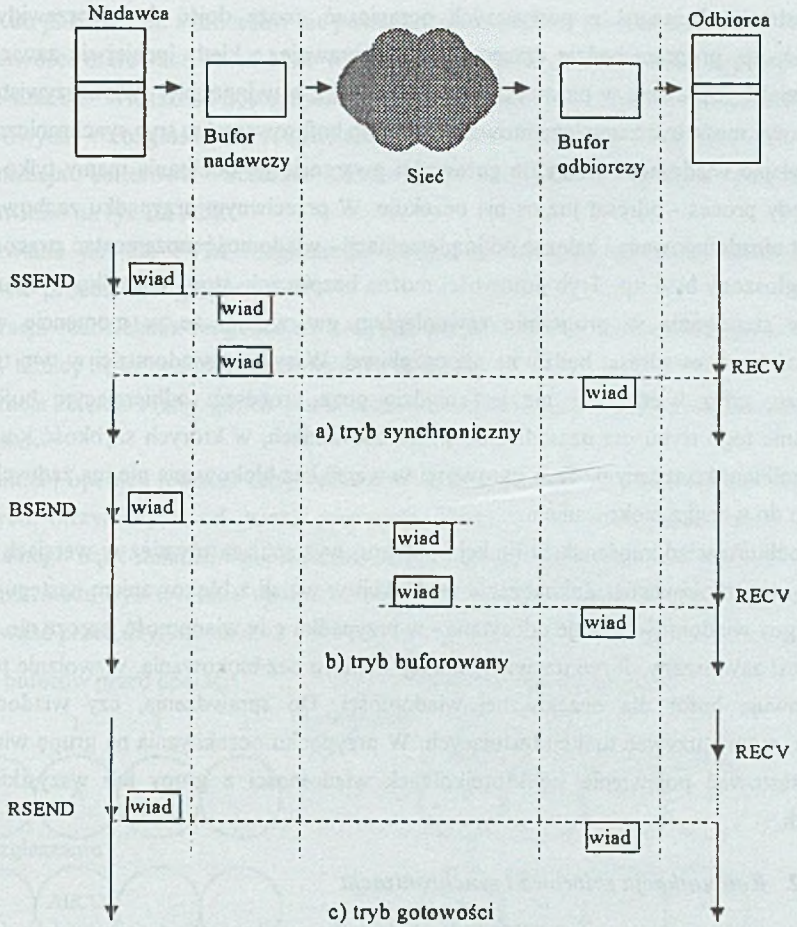
Opisana w poprzednim rozdziale, charakterystyczna dla biblioteki PVM,koncepcja maszyny wirtualnej powoduje, że na polu organizacji przetwarzania równoległego PVM przewyższa MPI pod względem oferowanych możliwości. Zupełnie inaczej sytuacja wygląda pod względem możliwości w zakresie przesyłania wiadomości.

4.3.1. Komunikacja punkt-punkt

PVM oferuje jeden tryb przesyłania wiadomości. Funkcja wysyłająca nie czeka na potwierdzenie odbioru wiadomości, ale kończy swoje działanie natychmiast po umieszczeniu wiadomości w sieci i zwolnieniu bufora do ponownego użycia. Wiadomości, które nie mogą być odebrane od razu, buforowane są po stronie odbiorcy. PVM zachowuje porządek wiadomości - jeżeli pewien proces wysłał do innego procesu kilka wiadomości, są one odbierane w takiej samej kolejności, w jakiej zostały wysłane.

Wiadomości są etykietowane definiowanymi przez użytkownika liczbami całkowitymi. Wiadomości do odczytu można wybierać według adresu nadawcy bądź według etykiety.

Standard MPI przewiduje cztery tryby komunikacji: synchroniczny, buforowany, standardowy i gotowości. Tryby te różnią się wykorzystaniem buforów - nadawcy i sieciowych. Sposób działania komunikacji w każdym z tych trybów pokazano na rys. 1.



Rys. 1. Tryby komunikacji punkt-punkt w MPI
Fig. 1. MPI communication modes

Z faktu zakończenia czynności wysyłania w **trybie standardowym** nie można wnioskować o odczytaniu bądź o nieodczytaniu wiadomości przez adresata. Przykładowo dwa procesy nie powinny stosować standardowego trybu komunikacji do wymiany między sobą dwóch wiadomości, gdyż może to w pewnych sytuacjach doprowadzić do wzajemnej blokady (podobna sytuacja może wystąpić w trybie synchronicznym). Podobnie proces nie powinien w tym trybie wysyłać kolejnych wiadomości, których interpretacja może być zależna od odebrania (przez inny proces) poprzednich wiadomości - podobnie jak w trybie buforowanym. Dodatkowo używając trybu standardowego należy przestrzegać tej samej zasady, co dla trybu

buforowanego - procesy muszą zapewniać odczytanie wszystkich wiadomości. Jeżeli program nie przestrzega któregoś z powyższych ograniczeń, może dojść do nieprzewidywalnych zachowań, np. program będzie czasem działał poprawnie, a kiedy indziej się zawieszał, lub będzie działał poprawnie w pewnej implementacji MPI, a w innej nie. W rzeczywistości tryb standardowy może być zaimplementowany jako tryb buforowany lub tryb synchroniczny.

Wysyłając wiadomość w **trybie gotowości** gwarancję jej odebrania mamy tylko w przypadku, gdy proces - adresat już na nią oczekuje. W przeciwnym przypadku zachowanie systemu jest niezdefiniowane i zależne od implementacji - wiadomość może zostać stracona, może zostać zgłoszony błąd itp. Tryb gotowości można bezpiecznie stosować tylko w sytuacji, gdy przepływ sterowania w programie równoległym gwarantuje, że w momencie wysyłania wiadomości proces-adresat będzie na nią oczekiwał. Wysyłanie wiadomości w tym trybie jest najszybsze, gdyż wiadomość nie jest nigdzie poza procesem odbierającym buforowana. Stosowanie tego trybu ma uzasadnienie w zastosowaniach, w których szybkość komunikacji jest czynnikiem krytycznym. Tryb gotowości w wersji bez blokowania nie ma żadnych zalet w stosunku do wersji z blokowaniem.

Do odbioru wiadomości służy funkcja odbioru, występująca również w wersjach z blokowaniem i bez blokowania. Zakończenie tej funkcji w wersji z blokowaniem następuje w momencie, gdy wiadomość zostaje odczytana - w przypadku gdy wiadomość jeszcze nie nadeszła, proces jest zawieszany. Jeżeli używamy funkcji odbioru bez blokowania, wywołanie tej funkcji przygotowuje bufor dla oczekiwanej wiadomości. Do sprawdzania, czy wiadomość już nadeszła, można używać funkcji testujących. W przypadku oczekiwania na grupę wiadomości można testować pojawienie się którejkolwiek wiadomości z grupy lub wszystkich oczekiwanych.

4.3.2. Komunikacja zbiorowa i synchronizacja

Istotnym elementem oceny bibliotek do przesyłu komunikatów jest ocena mechanizmów do komunikacji grupowej, umożliwiających, za pomocą jednego odwołania do biblioteki, wysłanie komunikatu do wielu procesów, odebranie komunikatów od wielu procesów lub inną formę synchronizacji. PVM oferuje tu użytkownikowi dwie funkcje: rozgłoszenie wiadomości (ang. *broadcast* lub *multicast*) i synchronizację barierową (ang. *barrier synchronization*).

Rozgłoszenie umożliwia wysłanie wiadomości do wszystkich procesów w grupie.

Synchronizacja barierowa polega na tym, że proces natrafiający na funkcję synchronizacji zostaje zawieszony w oczekiwaniu, aż określona w wywołaniu funkcji liczba procesów również wykona tę funkcję. W chwili, gdy ostatni z procesów wykona funkcję synchronizacji barierowej, wszystkie oczekujące procesy są odwieszane. Synchronizacja taka może mieć zastosowanie na przykład w sytuacji, kiedy pewien proces lub procesy oczekują, aż grupa

procesów zapisze do pliku wyniki swojego działania. Dopóki ostatni z wyników nie zostanie zapisany do pliku, żaden z procesów nie powinien tych wyników przetwarzać.

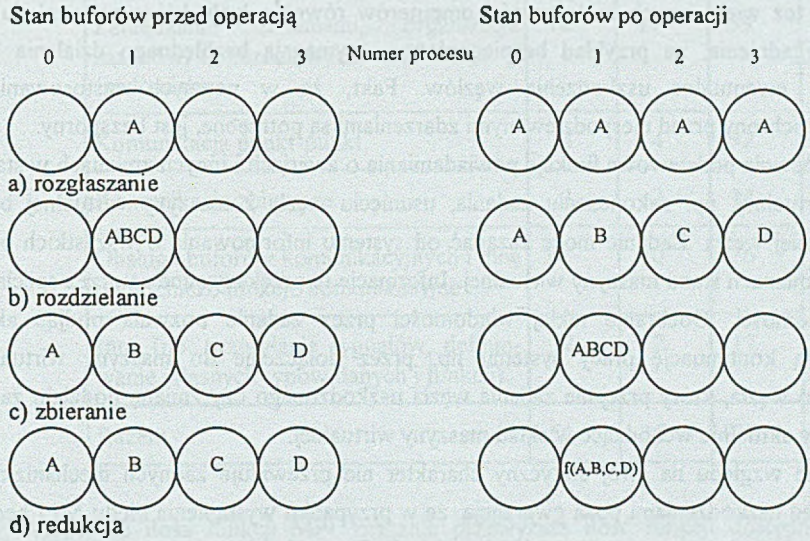
Możliwości oferowane przez MPI w zakresie komunikacji zbiorowej i synchronizacji są znacznie szersze. Większość dostępnych funkcji - to odmiany i kombinacje czterech operacji podstawowych - rozgłoszenia, rozdzielania, zbierania i redukcji. Dostępna jest również synchronizacja barierowa. Schemat działania poszczególnych operacji podstawowych przedstawiono na rys. 2a - 2d.

Omawiane już dla PVM rozgłoszenie polega na wysłaniu tej samej wiadomości do wszystkich procesów w grupie.

Operacja rozdzielania (ang. *scatter*) wysyła do procesów o kolejnych numerach kolejne elementy tablicy będącej argumentem wywołania funkcji.

Operacja zbierania (ang. *gather*) umieszcza w kolejnych polach tablicy wartości otrzymane od procesów.

Wynikiem operacji redukcji (ang. *reduction*) jest liczba będąca wynikiem pewnej operacji na danych otrzymanych od wielu procesów. Operacją tą może być wybór elementu maksymalnego bądź minimalnego, wartość średnia, suma odebranych wielkości itp. Operacja redukująca może być wybrana spośród dostępnych operacji standardowych bądź może być zdefiniowana przez użytkownika.



Rys. 2. Operacje komunikacji zbiorowej
Fig. 2. Collective communication operations

Silnym i wygodnym narzędziem zawartym w MPI są tzw. dziedziczone typy danych (ang. *derived datatypes)*. Standardowe typy danych umożliwiają wysłanie za pomocą MPI zmiennych prostych oraz tablic zmiennych takiego samego typu. W przypadku gdy chcemy wysyłać bardziej złożone struktury danych, musimy wykonać kilka operacji wysyłania lub skopiować odpowiednie wartości do zmiennej tymczasowej i wysłać tę właśnie zmienną. W wielu przypadkach nie jest to jednak konieczne dzięki możliwości skorzystania z definiowanych przez użytkownika, tzw. dziedziczonych typów danych. Umożliwiają one w szczególności definiowanie zmiennych nie rozmieszczonych w pamięci w sposób ciągły (np. kolumna tablicy dwuwymiarowej w sytuacji, gdy w pamięci dane rozmieszczane są kolejno wierszami), definiowanie zmiennych złożonych ze zmiennych prostych różnych typów rozmieszczonych w pamięci w sposób ciągły, a nawet zmiennych złożonych ze zmiennych prostych różnych typów nie rozmieszczonych w pamięci w sposób ciągły ani regularny.

5. Odporność na błędy i uszkodzenia

Odporność oprogramowania równoległego na uszkodzenia węzłów obliczeniowych jest zagadnieniem krytycznym w obliczeniach dużej skali, często trwających wiele godzin, a nawet dni. Istnieje też wiele innych zastosowań komputerów równoległych, które ze względu na charakter zagadnienia, na przykład bezpieczeństwo, wymagają bezbłędnego działania bez względu na ewentualne uszkodzenia węzłów. Fakt, że w pewnych zastosowaniach mechanizmy ochrony przed niespodziewanymi zdarzeniami są potrzebne, jest bezsporny.

PVM zapewnia podstawowe funkcje powiadamiania o awariach i innych zmianach w stanie maszyny wirtualnej, np. zakończeniu zadania, usunięciu węzła z maszyny wirtualnej bądź dodaniu do niej węzła. Zadanie może zażądać od systemu informowania o wszystkich bądź niektórych zmianach stanu maszyny wirtualnej. Informacje te przekazywane są przez specjalny rodzaj wiadomości. Odebranie takiej wiadomości przez zadanie pozwala podjąć akcję umożliwiającą kontynuację pracy systemu np. przez dołączenie do maszyny wirtualnej dodatkowego węzła, który przejmie zadania węzła uszkodzonego czy zmianę podziału zadań między węzły aktualnie wchodzące w skład maszyny wirtualnej.

MPI 1 ze względu na swój statyczny charakter nie przewiduje żadnych mechanizmów ochrony przed uszkodzeniami poza gwarancją, że w przypadku wystąpienia błędu MPI będzie możliwe zakończenie procesu (proces się nie zawiesi). Ze względu na wagę tego problemu do specyfikacji MPI 2 zostaną włączone podobne jak w PVM mechanizmy informowania o zdarzeniach specjalnych. Jest to konieczne, gdyż MPI 2 nie będzie już całkowicie statyczny, lecz będzie miał możliwość tworzenia i usuwania procesów podczas działania aplikacji MPI.

6. Zamiast podsumowania

Dotychczasowe rozważania koncentrowały się na ogólnym, jakościowym opisie obu konkurencyjnych bibliotek. Dla kompletności obrazu przedstawić należy jeszcze przynajmniej wybrane porównania ilościowe. Takie porównania dotyczyć mogą różnych cech. Poniżej zostaną przedstawione zestawienia, które wydają się być dla potencjalnego użytkownika najbardziej interesujące, mianowicie zestawienie ilości funkcji obu bibliotek z podziałem na kategorie oraz porównanie wydajności różnych implementacji PVM i MPI pracujących w różnych środowiskach sprzętowych.

6.1. Ilość funkcji PVM i MPI

W tabeli 1 zestawiono ilość funkcji obu bibliotek z podziałem na kategorie zadań, które dane funkcje wykonują.

Tabela 1

Ilość funkcji dostępnych w bibliotekach PVM i MPI z podziałem na kategorie

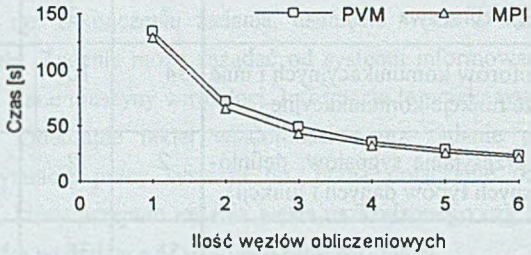
	PVM		MPI
	C	Fortran	
Zarządzanie zadaniami, organizacja zasobów obliczeniowych, informacje o środowisku	18	17	59
Komunikacja punkt-punkt,	4	4	32
Komunikacja zbiorowa	3	3	14
Obsługa buforów komunikacyjnych i inne pomocnicze funkcje komunikacyjne	34	10	6
Inne (np. przesyłanie sygnałów, definiowanie własnych typów danych i funkcji)	2	2	17
Razem	51	36	128

Widać, że ilość funkcji MPI znacznie przewyższa ilość funkcji dostępnych w PVM. Oznacza to, że możliwości oferowane przez MPI są znacznie szersze, jednak ze względu na niewielką ilość funkcji PVM biblioteka ta jest znacznie łatwiejsza do opanowania i sprawnego posługiwania się nią. Może to być zwłaszcza istotne dla użytkowników o słabym przygotowaniu informatycznym szukających łatwego w użyciu narzędzia do tworzenia

aplikacji rozproszonych. Z drugiej strony MPI oferując bardziej wyszukane możliwości komunikacyjne ułatwia pisanie złożonego oprogramowania, zaś jego funkcje, z których znaczną część trzeba by zastąpić skomplikowaną sekwencją wywołań funkcji PVM, pozwalają tworzyć kod zwarty i przejrzysty. Do zwartości i przejrzystości kodu MPI przyczynia się też prawie całkowity brak konieczności stosowania pomocniczych funkcji komunikacyjnych, zwłaszcza funkcji do zarządzania buforami (jawna obsługa buforów konieczna jest tylko w buforowanym trybie komunikacyjnym). Dzięki temu w wielu przypadkach sekwencję dwóch lub trzech wywołań PVM można zastąpić jednym wywołaniem MPI.

6.2. Porównanie wydajności PVM i MPI

Standard MPI został zaprojektowany tak, aby było możliwe stworzenie najbardziej wydajnych w danym środowisku implementacji. Implementacje MPI powinny więc pod względem szybkości działania przewyższać inne oprogramowanie realizujące przesył komunikatów, w tym bibliotekę PVM. Czy tak jest rzeczywiście, można ocenić przeprowadzając badania wydajności implementacji MPI i PVM w tym samym środowisku. Takie badania były przeprowadzane w kilku ośrodkach naukowych, m.in. w University of Texas (UT) [21], w NASA Lewis Research Center (LERC) [4] i w Cameron University (CU) [7]. Wszystkie testy przeprowadzone były w sieciach stacji roboczych. Testy UT i LERC koncentrowały się na badaniu szybkości i opóźnień przesyłu danych pomiędzy dwoma komputerami, zaś w teście CU przebadano konkretną aplikację (problem jednowymiarowej dyfuzji cieplnej) pracującą na zmiennej ilości węzłów. W teście CU implementacja MPICH MPI osiągnęła niewielką przewagę nad ORNL PVM, co pokazano na rysunku 3.

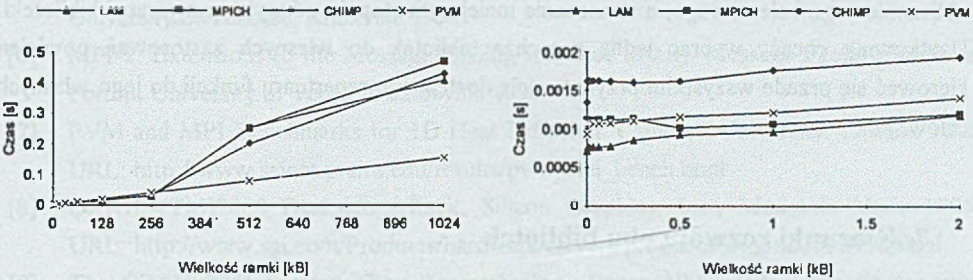


Rys. 3. Czas wykonania obliczeń dla aplikacji rozwiązującej problem jednowymiarowej dyfuzji cieplnej z wykorzystaniem bibliotek PVM i MPI (test CU)

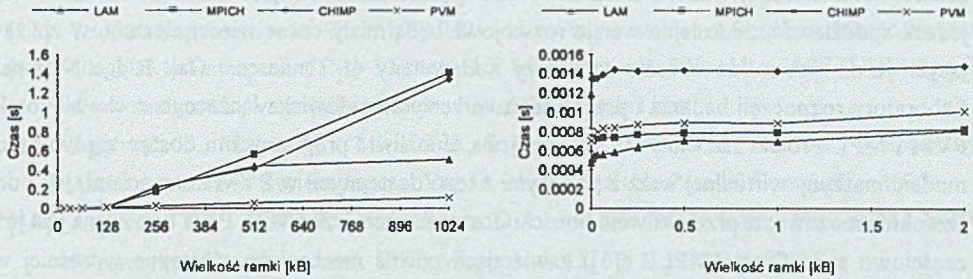
Fig. 3. Computation time for application solving 1-D Heat Diffusion Problem with use of PVM and MPI (CU test)

Najbardziej wszechstronny charakter miały testy LERC. Przebadano PVM w implementacji ORNL (podstawowa dostępna publicznie implementacja) oraz trzy publiczne implementacje MPI - LAM, MPICH i CHIMP. Do testów użyto komputerów IBM RS/6000 w różnych

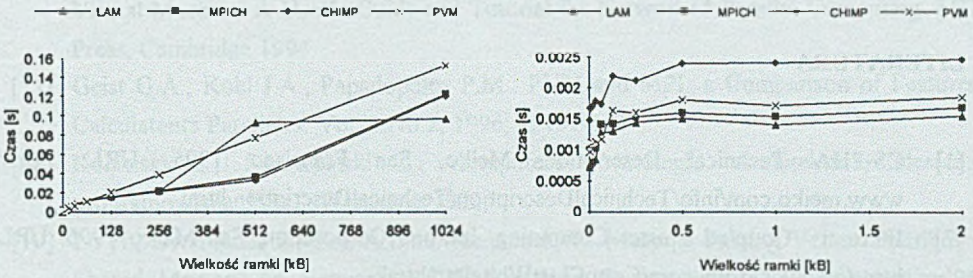
konfiguracjach środowisk sieciowych. Poniżej przedstawiono wykresy przedstawiające czas przesyłu wiadomości w zależności od wielkości tej wiadomości, uśredniony ze 100 prób, dla różnych bibliotek i różnych środowisk sieciowych. Dla każdego testu przedstawiono wyniki dla dużych i małych wielkości wiadomości. Na wykresach tych widać, że żadna z bibliotek nie osiąga zdecydowanej przewagi, zaś wyniki osiągnięte przez poszczególne implementacje zależą w dużym stopniu od środowiska oraz od wielkości przesyłanych wiadomości.



Rys. 4. Czasy przesyłu wiadomości w środowisku z siecią FDDI (test LERC)
Fig. 4 Message transfer times in FDDI network environment (LERC test)



Rys. 5. Czasy przesyłu wiadomości w środowisku z siecią ATM (test LERC)
Fig. 5. Message transfer times in ATM network environment (LERC test)



Rys. 6. Czasy przesyłu wiadomości w środowisku z siecią FCS (test LERC)
Fig. 6. Message transfer times in FDDI network environment (LERC test)

W teście UT przebadano tylko implementacje MPICH MPI oraz ORNL PVM. Testy wykonano na komputerach Sun, w środowiskach sieciowych ATM i Ethernet. Wyniki były korzystniejsze dla MPI, jednak różnice nie były znaczące. Również tutaj autor zwraca uwagę na dużą zależność wyników od środowiska komunikacyjnego, nie tylko sprzętowego, ale też programowego (np. implementacja protokołów IP na ATM).

Podsumowując wyniki wszystkich testów porównujących PVM z MPI można zauważyć, że różnice w wydajności obu bibliotek zależą w dużym stopniu od wyboru środowiska obliczeniowego i sieciowego, a w znacznie mniejszym stopniu od wykorzystywanej biblioteki. Użytkownik chcący wybrać jedną z tychże bibliotek do własnych zastosowań powinien kierować się przede wszystkim przydatnością dostępnego repertuaru funkcji do jego własnych celów.

7. Kierunki rozwoju obu bibliotek

Wydaje się, że obie przedstawione w artykule biblioteki, umożliwiające oparte na paradygmacie przesyłu wiadomości, zawierają unikalne, przydatne w określonych zastosowaniach cechy i żadna z nich w najbliższym czasie nie wyjdzie z użycia. Można się jednak spodziewać, że kolejne wersje rozwojowe będą miały coraz więcej elementów zbliżających je do siebie. Już obecnie naukowcy z University of Tennesee i Oak Ridge National Laboratory rozpoczęli badania i prace nad stworzeniem środowiska łączącego cechy bibliotek PVM i MPI. Projekt nazwany PVMPI [15] ma umożliwić programistom dostęp zarówno do modelu maszyny wirtualnej wraz z płynącymi z tego dostępnymi w PVM korzyściami, jak i do szerokich możliwości przesyłu wiadomości udostępnianych przez MPI. Poza tym znana jest już częściowo specyfikacja MPI 2 ([6]) zawierająca pewne mechanizmy dostępne wcześniej w PVM, a nieobecne w MPI 1. Należy również przewidywać, że specyfikacja MPI nie pozostanie bez wpływu na kolejne wersje PVM, jeżeli takie powstaną.

LITERATURA

[1] CS-2HA Technical Description. Meiko, San Francisco 1995. URL: http://www.meiko.com/info/TechnicalDescription/TechnicalDescription.html

[2] Inktomi - Coupled Cluster Computing. Inktomi Corporation, San Mareo 1996. URL: http://www.inktomi.com/CoupClustWhitePap.html

[3] Metacomputing: A Reality Check. National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign 1995. URL: http://www.ncsa.uiuc.edu/Cyber-ia/Comp/MetaReality.html.

[4] MPI and PVM benchmarks on the LACE cluster. NASA Lewis Research Center, Cleveland, URL: http://www.lerc.nasa.gov/WWW/ACCL/PARALLEL/TIMING/benchmark.html

[5] MPI: A Message Passing Interface Standard. Message Passing Interface Forum, University of Tenesee, Knoxville 1994.

[6] MPI-2: Extensions to the Message Passing Interface (draft). Message Passing Interface Forum, University of Tenesee, Knoxville 1996.

[7] PVM and MPI Benchmarks for 1D Heat Diffusion. Cameron University, Lawton 1996 URL: http://www.scinet.praire.edu/results/pvm_mpi_bench.html

[8] ORIGIN(TM)2000 Deskside &Rack. Silicon Graphics, Inc., Mountain View 1996. URL: http://www.sgi.com/Products/hardware/servers/products/Origin2000Desk.html

[9] The CRAY T3D System. Cray Research, Inc., Eagan 1997. URL: http://www.cray.com/PUBLIC/product-info/mpp/CRAY_T3D.html

[10] The CRAY T3E Scalable Parallel Processing System. Cray Research, Inc., Eagan 1997. URL: http://www.cray.com/PUBLIC/product-info/T3E/CRAY_T3E.html

[11] Top500 Supercomputer Sites. 1996 URL: http://www.netlib.org/benchmark/top500.html

[12] Ultra(TM) Enterprise (TM) 10 000. Sun Microsystems. Palo Alto 1997. URL: http://www.sun.com/servers/datacenter/products/starfire/index.html

[13] Agerwala T., Martin J.L., Mirza J.H., Sadler D.C., Dias D.M., Snir M.: SP2 System Architecture IBM System Journal Vol.34, No.2 - Scalable Parallel Computing, 1995

[14] Anderson T.E., Culler D.E., Patterson D.A. and the NOW team: A Case for NOW (Networks of Workstations). IEEE Micro. Vol. 15, No. 1, 1995, s. 54-64.

[15] Fagg G.E., Dongarra J.: PVMPI: An integration of the PVM nad MPI systems. University of Tenesee technical report ut-cs-96-328, Konxville 1996.

[16] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderam V.: PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge 1994.

[17] Geist G.A., Kohl J.A., Papadopulos P.M.: PVM and MPI: a Comparison of Features. Calculateurs Paralleles. Vol. 8 No.2, 1996, ss.137-150

[18] Kozielski S., Szczerbiński Z.: Komputery równoległe. Architektura, elementy programowania. WNT, Warszawa 1993.

[19] Lu H., Dwarkadas S., Cox A.L., Zwaenepoel W.: Message Passing Versus Distributed Shared Memory on Networks of Workstations. Proceedings of Supercomputing'95, Urbana-Champaign 1995.

[20] MacDonald N., Minty E., Harding T., Brown S.: Writing Message-Passing Programs with MPI. Course Notes. Edinburgh Parallel Computing Centre, Edinburgh 1996.

[21] Zuazaga H.O.: A performance comparison of MPICH and PVM over Ethernet and ATM networks. University of Texas, San Antonio, 1996. URL: http://momo.uth-scsa.edu/~humberto/class/dist/mpi-vs-pvm.html

Recenzent: dr inż. Katarzyna Stąpor

Wpłynęło do Redakcji 8 września 1997 r.

Abstract

The article compares two standards of message-passing library: PVM and MPI. PVM is more flexible, easy reconfigurable (also during computations) and failure-proof environment. PVM also offers higher level of interoperability. MPI has more communication functions, among them multiple communication modes and rich set of collective communication functions, but due to it's static character it is not fitted to use in failure-proof applications. MPI should offer higher performance than PVM, but as it was shown during various tests, this expectation is not always fulfilled.