Politechnika Śląska w Gliwicach

Obliczenia Równoległe II

Komunikacja między procesami oraz redukcja danych (MPI)

Autor:
Bartłomiej Buchała

Informatyka SSM, semestr II
Rok akademicki 2016/2017
Grupa OS1

8 listopada 2016

1 Wstęp

Wraz z rozwojem nauk ścisłych pojawiają się coraz bardziej skomplikowane problemy natury naukowej. Do ich rozwiązania niezbędne są komputery o dużej mocy obliczeniowej. Rozwój technologii pozwolił jednak na stworzenie maszyn o większej szybkości obliczeń. Zgodnie z prawem Moore'a, które zakłada, że liczba tranzystorów w procesorach rośnie wykładniczo, moc obliczeniowa jrdnostek centralnych wzrasta dwukrotnie co każde dwa lata. Przy stałym zwiększaniu taktowania procesora, napotkano jednak pewien problen – dla pewnego progu zużycie prądu (a przede wszystkim temperatura pracującego CPU) rosną eksponencjalnie w stosunku do częstotliwości taktowania. W okolicach 2005 roku, większość producentów procesorów zdecydowała się wykorzystać inne podejście – zastosować obliczenia równoległe. W tym celu, zamiast rozwijać coraz to szybsze (a konkretnie – wyżej taktowane) procesory monolityczne, kolejne jednostki centralne miały zostać wyposażone w wielokrotne procesory zintegrowane w jednym obwodzie – tak zwane **procesory wielordzeniowe**.

Dodanie dodatkowych rdzeniów nie rozwiązało jednak w magiczny sposób problemów z wydajnością w przypadku większości istniejących programów. Główną przyczyną był fakt, że spora część algorytmów przygotowana była z myślą o wykonaniu sekwencyjnym – czyli przeznaczonym do wykonaniu na jednym procesorze. W tym przypadku wykonywany kod nie był świadomy obecności innych jednostek obliczeniowych, co uniemożliwiało ich użycie przy wykonywaniu kolejnych rozkazów. Szybkość z jaką wykonywał się taki program była zazwyczaj zbliżona do tej wyliczonej w trakcie użycia jednego procesora. Do doprowadziło do powstania do programów równoległych.

Przez **programowanie równoległe** rozumiemy taką motodę tworzenia algorytmu, która pozwala jednoznacznie wskazać, które fragmenty obliczeń mają zostać wykonane w sposób równoległy na osobnych procesorach. W tym celu wyodrębniono 3 pojęcia:

Program współbieżny (ang. *concurrent*) występuje w przypadku, gdy procesy są wykonywane przez jeden procesor rzeczywisty metodą przeplotu.

Program równoległy (ang. *parallel*) to przypadek, kiedy każdy proces wykonywany jest przez osobną jednostkę obliczeniową, a procesory posiadają dostęp do wspólnej pamięci.

Program rozproszony (ang. *distributed*) występuje, gdy procesy wykonywane są przez odrębne, rozproszone procesory połączone kanałami komunikacyjnymi.

W pierwszym rozpatrywanym przypadku nie dochodzi do prawdziwego wykonania równoległego, ponieważ w dowolnym momencie czasu nie istnieją przynajmniej 2 procesy, które są wykonywane jednocześnie. Obliczeniami zajmuje się jeden procesor, a kolejne rozkazy procesów wykonywane są na zmianę – pomiędzy nimi zachodzi przełączanie kontekstu (zapamiętanie niezbędnych danych dotyczących stanu procesu). W dwóch pozostałych scenariuszach, należy rozwiązać dodatkowo jeden problem: komunikację między procesorami w określonych momentach obliczeń. Istnieją dwa so=posoby realizacji takiego przedsięwzięcia:

- Wykorzystanie **pamięci wspólnej** – zakłada ono istnienie pamięci operacyjnej, w której znajdują się dane potrzebne do obliczeń. Każdy procesor biorący udział w obliczeniach ma dostęp do zmiennych wspólnych (ang *shared variables*), na których może wykonywać określone operacje (np. operacje czytania, zapisu, lub bardziej zaawansowane jak porównanie-zamiana lub czytanie-modyfikacja-zapis).

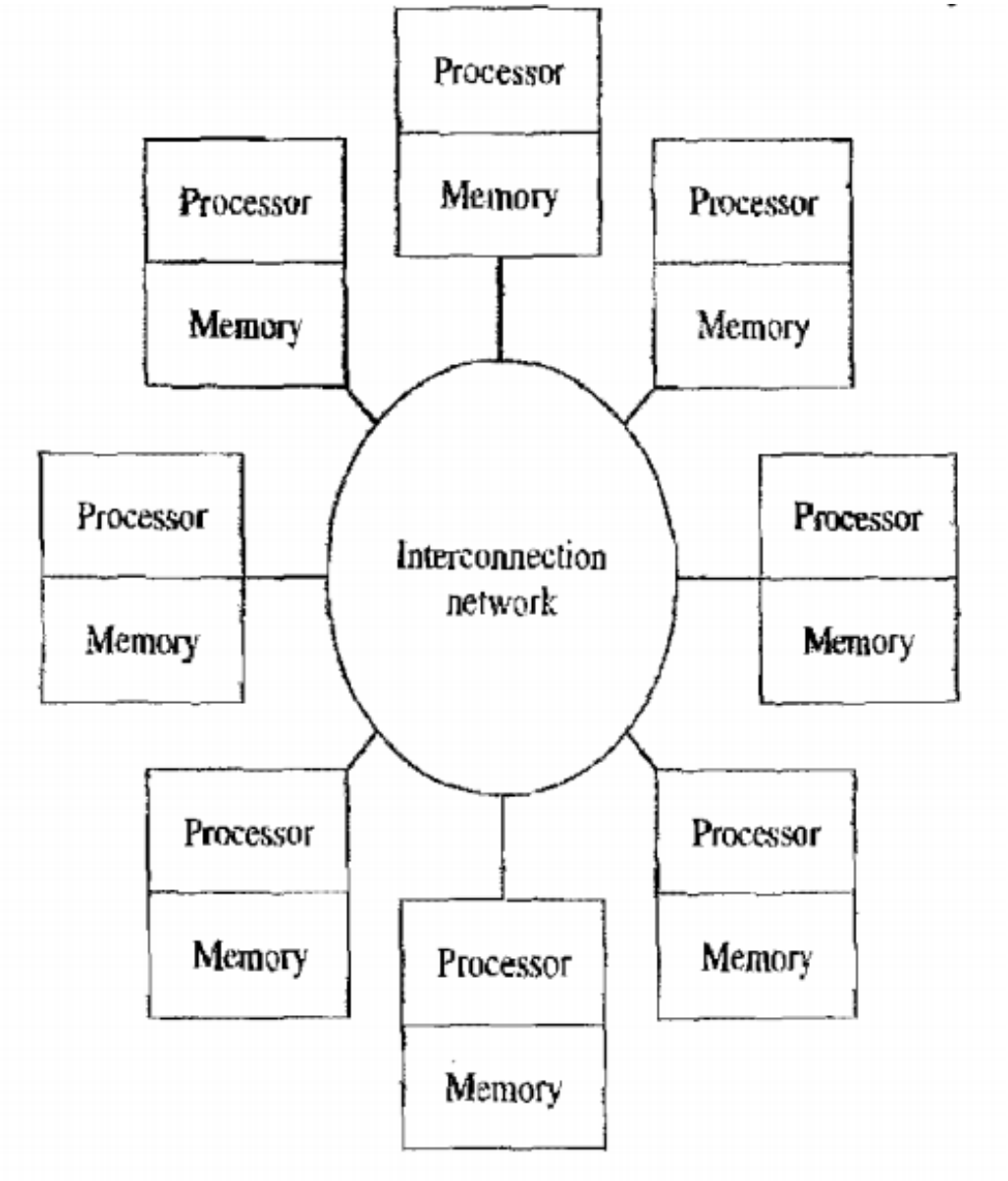
- Przesył wiadomości za pomocą **kanałów komunikacyjnych** – zakłada istenie specjalnych kanałów, poprzez które procesory wysyłają między sobą wirtualne wiadomości. Każdy kanał jest dwukierunkowy i łączy 2 procesory, natomiast ich zbiór stanowi sieć połączeń. Procesory w klastrze mogą był połączone w różne schaematy, np. listy cyklicznej czy macierzy. Obliczenia wykonywane są asynchronicznie, gdyż nie można dokładnie określić momentów, w których operacje wykonywane są współbieżnie, a także momentów wysyłu i odbioru wiadomości pomiędzy poszczególnymi CPU.

W latach 90-tych XX wieku powstały dwa standardy, które miały ułatwić tworzenie i pracę z kodem przeznaczonym do wykonania równoległego: OpenMP (ang. *Open Multi-Processing*), który charakteryzuje się wykorzystaniem pamięci wspólnej) oraz MPI (ang. *Message Passing Interface*), korzystający z kanałów komunikacyjnych. Dalsza część referatu zostanie poświęcona temu drugiemu.

2 Interfejs MPI

2.1 Model sieciowy

Komunikacja odbywa sie za pomocą przesyłania wiadomości (czyli między innymi w standardzie MPI)w tak zwanym modelu sieciowym. Składa się on z określonej liczby procesorów, przy czym każdy z nich posiada własną pamięć lokalną. Procesory posiadają dostęp jedynie do instrukcji i danych przechowywanych w swojej pamięci lokalnej – nie istnieje pamięć wspólna. Aby umożliwić wymianę informacji pomiędzy procesorami, tworzona jest sieć połączeń (ang *interconnection network*), która zbudowana jest z dwukierunkowych kanałów komunikacyjnych (łącz).



Rysunek 1: Model sieciowy. Źródło: [4], str 94

Wymiana informacji między procesorami jest realizowana poprzez kooperujące ze sobą procedury trasowania (ang. *routing*), które działają w każdym procesorze. Dzięki nim, każdy węzeł sieci (tutaj: procesor) posiada informację, z którymi węzłami może wymieniać informacje. Zbiór wszystkich procedur trasowania definiuje **topologię sieci połączeń**. Można ją opisać przy pomocy grafu, gdzie wierzchołkami (węzłami) są procedory, natomiast krawędzi to dwukierunkowe łącza.

Ocenę skuteczność/przydatność danej sieci podczas prowadzenia obliczeń równoległych można określić biorąc pod uwagę kilka parametrów:

- **Średnica sieci** (ang *diameter*) – maksymalna odległość zmierzoną za pomocą liczby krawędzi między dowolnymi dwoma wierzchołkami. Im mniejsza średnica, tym lepsza jest sieć – oznacza to, że informacje będą potrzebowały średnio mniej czasu na dotarcie do właściwego odbiorcy. Przypadek pesymistyczny zakłada, że wiadomość będzie musiała zostać przesłana przez liczbę krawędzi równej średnicy.

- **Szerokość połowienia sieci** (ang. *bisection width*) – minimalna liczba krawędzi, którą należy usunąć z obecnej sieci, aby móc ją podzielić na 2 równe podsieci.

- **Szerokość pasma** (ang. *bisection bandwidth*) – jest to iloczyn szerokości połowienia oraz szybkości przesyłu danych w pojedynczym kanale. Pozwala określić liczbę bitów, jaką można przesłać między połówkami w jednostce czasu. Im większa szerokość pasma, tym lepiej.

- **Maksymalny stopień wierzchołka** – maksymalna liczba krawędzi połączonych z danym wierzchołkiem (liczona globalnie dla całej sieci). Dla niewielkiego stopnia łatwiej zaprogramować procedury komunikacyjne ze względu na fakt, że używają one mniejszej liczby kanałów. Zakłada się, że sieć jest dobra jeżeli przy wzroście liczby p procesorów średnica sieci rośnie nie szybciej niż logarytmicznie w funkcji p, natomiast maksymalny stopień wierzchołka jest stałą liczbą o małej wartości.

- **Spójność krawędziowa** (ang. *edge connectivity*) – definiowana jaka minimalna liczba krawędzi, które muszą zostać wyłączone z sieci aby ta stała się niespójna (graf rozłoży się na 2 lub więcej osobnych podgrafów). Im większa spójność krawędziowa, tym odporniejsza jest sieć – istnieje mniejsze prawdopodobieństwo całkowitego unieruchomienia sieci w przypadku, gdy któryś procesor ulegnie uszkodzeniu. Większa spójność prowadzi też do zmniejszenia rywalizacji poszczególnych węzłów o łącze.

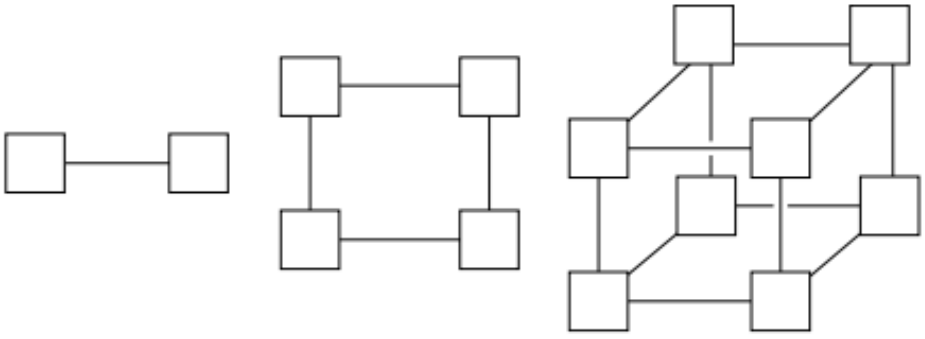
- **Koszt sieci** – zazwyczaj określana jako suma wszystkich kanałów w sieci.

Przykładowe topologie sieci połączeń:

- siatka

- torus (jedno-, wielowymiarowy)

- kostka (jedno-, wielowymiarowa)



Rysunek 2: Topologia kostki. Od lewej: jedno-, dwu- oraz trójwymiarowa. Źródło: [3], str 40

Literatura

[1] Z. J. Czech, *Wprowadzenie do obliczeń równoległych*, Wydawnictwo PWN, Warszawa 2013, wyd. 2

[2] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard, Version 3.0*, High Performance Computing Center Stuttgart (HLRS), Stuttgart 2012

[3] P. Pacheco, *An Introduction to Parallel Programming*, Morgan Kaufmann, San Francisco 2001

[4] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill, Nowy Jork 2003