ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1995

Seria: INFORMATYKA z. 28 Nr kol. 1270

Krzysztof SCHIFF

EFEKTYWNOŚĆ PROJEKTOWANYCH SYSTEMÓW RÓWNOLEGŁYCH

Streszczenie. W pracy omówiono programowe narzędzie służące do badania architektur systemów wieloprocesorowych i eksperymentowania z takimi problemami jak projektowanie programów równoległych, harmonogramowanie procesów, grupowanie ich, obciążanie procesorów zadaniami oraz określanie polityki wyznaczania marszrut dla przesyłanych komunikatów pomiędzy procesorami. Omówiono bloki edycji programu, symulacji wykonania zadanego programu w zadanej architekturze i blok graficznej wizualizacji rezultatów symulacji systemu.

PERFORMANCE OF DESIGNING PARALLEL SYSTEMS

Summary. In this work a software tool is discribed that provides a multiprocessor system for research into alternative architectural decisions and experimentation, with such issues as: design parallel programs, scheduling, clustering, load processor balancing and routing policies. Also program editor, behaviour simulator block and visualization & evaluation tool is presented.

L'EFFECTIVITE DES PROJETES SYSTEMES PARALLELES

Résumé. Dans cet article, on presente l'outil programme qui sert a la recherche de l'archutecture du systems multiprocesseurs et a l'experimentation avec tel problems comme : le construction du programme parallel, du harmonogramme du processus, du groupage du processus, du chargement du processeurs et de la politique du routage. On est aussi decrit l'editor du programme, le segment du simulateur du comportement

de programme parallele pendant la realisation et le bloc de la visualisation des resultats de la simulation.

1. Wprowadzenie

Komputery równoległe różnią się między sobą wieloma cechami takimi jak np.: ilością procesorów i topologią połączeń. Z tego też względu opracowanych algorytmów dla określonych maszyn równoległych często nie udaje się przenieść na inne. W celu uruchomienia algorytmu na innej maszynie użytkownik musi przystosować przenoszony algorytm do nowej architektury komputera [6, 7]. Nawet w takich przypadkach, gdy algorytm udaje się przenieść, jego efektywność realizacji na obu komputerach z reguły odbiega od siebie. Potrzeba optymalnego zharmonizowania algorytmu równoległego z architekturą komputera wymusza korzystanie z programowych narzędzi, które umożliwią symulację przebiegu realizacji algorytmu i odpowiednie dopasowanie parametrów systemu równoległego, takich jak: stopień zgranulowania zadań, liczba procesorów, kolejność realizacji zadań, topologia połączeń procesorów, sposób wyznaczania marszrut komunikacyjnych, itp.

Przykładem zestawu narzędzi programowych służących do "zgrania" algorytmu z architekturą jest PSEE (Parallel System Evaluation Environment) [4]. Pozwala on na badanie zachowania się systemów równoległych o rozproszonej pamięci. Korzystając z niego można tworzyć algorytmy równoległe, symulować ich realizację na różnych architekturach, otrzymywać ocenę efektywności zachowania się systemu równoległego i zarządzać głównymi parametrami systemu równoległego. "Zgranie" algorytmu z architekturą przebiega cyklicznie poprzez programowanie, symulację, pomiar, wizualizację wyników i modyfikację parametrów systemu równoległego.

Artykuł ten jest zorganizowany w następujący sposób: rozdział drugi opisuje strukturę środowiska programowego, rozdział trzeci opisuje formalizm wykorzystywany przy modelowaniu systemu równoległego, rozdział czwarty opisuje podstawowe instrukcje edytora zawartego w PSEE i rozdział piąty wizualizację pomiarów głównych cech zachowania się systemu równoległego.

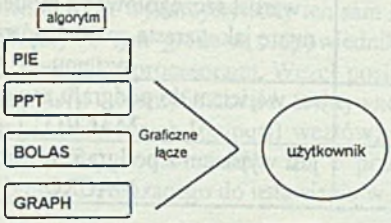
2. Struktura PSEE

Środowisko PSEE jest zbudowane z czterech głównych modułów:

— edytora PIE, który pozwala na napisanie źródłowego programu równoległego,
— translatora PPT, który przetwarza źródłowy program równoległy w sekwencyjny kod do wykonania na jednym procesorze,
— symulatora BOLAS, który modeluje program użytkownika i architekturę równoległą, przeprowadza symulację i generuje dane do późniejszej wizualizacji,
— bloku wizualizacji GRAPH, który wyświetla zbiory pomiarów głównych parametrów za-

chowania się systemu podczas realizacji programu równoległego.

Zestaw narzędzi zawartych w PSEE pozwala na rozwój intuicji programisty dotyczącej systemów równoległych. Symulator BOLAS [4,5] i blok wizualizacji GRAPH [3,4] umożliwiają projektantowi systemów równoległych dopasowanie programu użytkownika do ustalonego komputera równoległego, porównanie różnych algorytmów rozwiązujących ten sam problem, uwidocznienie wpływu architektury systemu równoległego na przebieg algorytmów, nabycie doświadczenia w projektowaniu algorytmów dla różnych typów architektur równoległych, itp..



Rys.1. Struktura PSEE
Fig.1. General block diagram of PSEE

3. Formalizm modelowania

3.1. Modelowanie programu równoległego

Formalizm użyty do modelowania w PSEE nazwany został WBG (weighted behavioural directed graph - dynamiczny graf skierowany z wagami) [4]. Elementami tego grafu są węzły i łuki. Węzły odpowiadają sekwencyjnym segmentom programu, łuki zaś reprezentują relacje zależności pomiędzy segmentami. Węzeł jest charakteryzowany przez dwa parametry "volume" (liczba instrukcji do wykonania) i "class" (definiuje zachowanie się węzła). Czas realizacji zadań węzła zależy od "volume" i efektywności procesora, któremu został przypisany. Parametr "class" opisujący zachowanie węzła może przynależeć do jednej z trzech kategorii klas:

1 kategoria - klasy związane z prostym modelowaniem kodu sekwencyjnego; przykładem jest tu klasa STANDARD.
2 kategoria - klasy związane z dynamicznym zachowaniem się WBG, pozwalają one na aktywowanie kopii węzłów; przykładami są: Call, Return i Join.
3 kategoria - klasy związane z hierarchiczną definicją WBG i ze złożonością jego zarządzania; przykładami są klasy: Macro, Input i Output.

Tabela 1

Klasy węzłów algorytmów i architektury

KLASY WĘZŁÓW	PRZEZNACZENIE
STANDARD	segment sekwencyjnego kodu
CALL	generuje nową pętlę pozwalającą na równoległą realizację podgrafu
RETURN	kończy realizację nowej pętli i powraca do realizacji starej
JOIN	modeluje wszystkie pętle lub zakończenie procedury rekursywnej
MACRO	pozwala stworzyć strukturę grafu i celowy wzrost szczegółowości modelu WBG, w miarę jak wzrasta znajomość rzeczywistego systemu
INPUT	jest wejściem do podgrafu zawartego w węźle MACRO
OUTPUT	jest wyjściem z podgrafu zawartego w węźle MACRO

Każdy węzeł posiada politykę wejścia (OR i AND) i politykę wyjścia (OR i AND). Polityka wejścia determinuje uruchomienie segmentu sekwencyjnego, zaś polityka wyjścia wyznacza łuki, którymi będzie przekazywana informacja. Informacja przepływająca poprzez graf jest modelowana za pomocą żetonów.

AND polityka oznacza, że węzeł (segment sekwencyjny) jest uruchamiany, gdy co najmniej jeden żeton jest obecny przy każdym wejściu do węzła. OR polityka oznacza, że węzeł jest uruchamiany, gdy żeton jest obecny przy którymkolwiek z wejść do węzła. Oczywiście kombinacje tych polityk są możliwe.

Węzeł jest opisywany w następujący sposób:

ni : Id, Vp, Class, Pin, Pout, [inputs], [outputs]

gdzie:

Id – identyfikator węzła,
Vp – liczba instrukcji do zrealizowania,
Class – klasa węzła,
Pin – polityka wejścia,
Pout – polityka wyjścia,
[inputs] – grupa węzłów po stronie wejścia,
[output] – grupa węzłów po stronie wyjścia.

Łuki są charakteryzowane poprzez "volume", czyli poprzez przepuszczaną przez siebie ilość żetonów. Czas komunikacji jest zależny od węzła źródła i przeznaczenia oraz algorytmu wyznaczającego marszrutę komunikacyjną.

Łuk jest opisywany w następujący sposób:

ai : Ids, Idd, Vc

gdzie:

Ids – identyfikator węzła źródłowego,
Idd – identyfikator węzła przeznaczenia,
Vc – ilość przepływających żetonów.

3.2. Modelowanie architektury

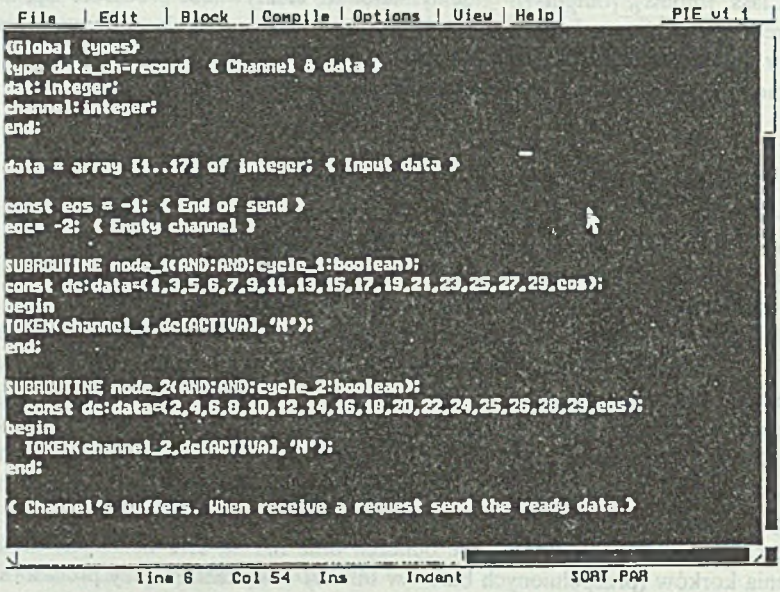
Do modelowania architektury jest wykorzystywany ten sam formalizm co do modelowania programu równoległego. Węzły w tym grafie są odpowiednikami procesorów, zaś łuki są odpowiednikami połączeń pomiędzy procesorami. Węzeł posiada następujące parametry (n: Id, Cp, Class, [inputs], [outputs]): identyfikator Id, efektywność procesora Cp, klasę Class, zespół wejściowych [inputs] i wyjściowych [outputs] węzłów). Łuk opisywany jest poprzez parametry (I : Ids, Idd, Cc, Bs): węzeł źródła Ids i przeznaczenia Idd, efektywność komunikacji Cc i wielkość bufora Bs służącego do ustawiania w kolejkę żetonów.

3.3. Funkcje systemowe

Oprócz modelowania programu i architektury symulator BOLAS wymaga określenia dwóch cech systemowych: kolejności wykonywania zadań i sposobu wyznaczania marszrut dla przesyłu komunikatów. Określenie kolejności wykonywania zadań ma na celu takie jej ustawienie, by czas realizacji wszystkich zadań na wszystkich dostępnych procesorach był jak najmniejszy. Uzyskuje się w ten sposób maksymalne wykorzystanie zasobów komputerowych i redukuje się komunikację pomiędzy procesorami. W PSSE można znaleźć trzy sposoby wyznaczania kolejności realizacji zadań na procesorach: sposób ręczny, korzystając z algorytmu ścieżki krytycznej (CP) oraz z algorytmu ścieżki krytycznej z natychmiastowym pierwszym następcą (CP/MISF) [1, 4]. Określenie, w jaki sposób będą przesyłane komunika-ty pomiędzy procesorami, ma na celu zmniejszenie udziału czasu komunikacyjnego w całości czasu potrzebnego na przeprowadzenie obliczeń oraz ma na celu uniknięcie możliwych do wystąpienia korków (przepełnionych buforów łuków) w łączach między procesorami. Węzły (procesory) w WBG komunikują się przekazując komunikaty (żetony), często przepływają one przez dużą część grafu. Komunikacja pomiędzy węzłami, które nie są bezpośrednio ze sobą połączone, realizowana jest poprzez pośredniczące w komunikacji procesory. Wybór najbardziej odpowiedniej drogi w danej chwili czasowej nazwany jest polityką wyznaczania marszrut komunikacyjnych. W PSEE można dokonać wyboru drogi komunikacyjnej na dwa sposoby: ręcznie lub poprzez algorytm wyznaczania minimalnej ilości pośrednich etapów komunikacyjnych dla architektury typu 3D hipersześcian [7].

4. Podstawowe instrukcje edytora PIE

By napisać program równoległy, wykorzystujemy interaktywny edytor PIE, następnie korzystając z PPT uzyskujemy wersję .exe programu, który może być wykonany w sposób sekwencyjny na jednym procesorze. Później możemy edytować model programu i architektury, przypisywać węzły programu do węzłów architektury, symulować działanie na nieograniczonej liczbie procesorów i połączeń przy zerowym czasie komunikacji lub symulować działanie na ograniczonej architekturze, wyznaczać trasy komunikacyjne i tworzyć do późniejszej wizualizacji zbiory zawierające dane o zachowaniu się systemu podczas realizacji programu.



Rys.2. Edytor PIE – edycja programu głównego
Fig.2. Editor PIE – the main program edition

Edytor PIE jest wyposażony w podstawowe instrukcje, które umożliwiają wyrażenie działań odbywających się jednocześnie i które są przedstawione poniżej.

I. Podstawową deklaracją jest SUBROUTINE, która składa się z nagłówka, stałych lokalnych, typów, deklaracji zmiennych i ciała podprogramu.

```
SUBROUTINE nazwa (input_policy; output_policy; param[param]: type);

    Const; Type; Var;
        {'lokalne procedury i funkcje'}
        Begin
        ...
        End;
```

input_policy - identyfikuje zachowanie się węzła ze względu na wejście (polityka and lub or),
output_policy - sposób rozchodzenia się żetonów po realizacji segmentu sekwencyjnego węzła (polityka and lub or),
param - identyfikatory zmiennych, które aktywują wykonanie węzła.

Przykład:

```
type data_channel = record {Globalny typ}
        dat     : integer;
        channel : integer;
        end;

Subroutine węzeł_3 (AND; AND; channel_1 : integer; request_1 : boolean);
var dc : data_channel; { lokalna zmienna}
  begin
      dc.dat:=channel_1;
      dc.channel:=1;
      TOKEN(data_c,dc,'N') {komunikacja z węzłem_5 – zobacz poniżej}
  end;
```

II. Komunikaty są przekazywane od jednego podprogramu do drugiego za pośrednictwem deklaracji żetonu:

```
TOKEN(parameter, expression, type);
```

parameter : identyfikator podprogramu przeznaczenia
expression : wartość żetonu
type : STANDARD, CALL lub RETURN

Przykład:

```
SUBROUTINE węzeł_7(AND; AND; sync_1 : integer);
  begin
      TOKEN(request_1, true, 'N');
```

```
            {Przekazuje komunikat do węzła ze zmienną wejściową request_1
(węzeł_3) i wartość : True}
        end;
```

III. Zmienna statyczna może być zadeklarowana w następujący sposób:

```
VARHOLD
            Begin
        var_nazwa[, var_nazwa] : type;
            End;
```

Przykład:

```
SUBROUTINE węzeł_5(OR; AND; data_c : data_channel);
    var pipe : integer;
    VARHOLD begin
        {Zachowuje static_1 i static_2 wartości w różnych realizacjach}
            static_1, static_2 : integer;
            end;

      Begin
            if data_c.channel = 1 then static_1:=data_c.dat
          else static_2:=data_c.dat;
        {Porównuje nową zmienną z poprzednią i wysyła komunikat}
          if static_1>static_2 then pipe:=1 else pipe:=2;
          TOKEN(następny_rep, pipe, 'N');
        end;
```

IV. ACTIVA zwraca liczbę, która określa, ile razy podprogram SUBROUTINE był wykonywany. Ten operator daje możliwość warunkowej realizacji n-razy węzła lub ustalenia lokalnej zmiennej.

Przykład:

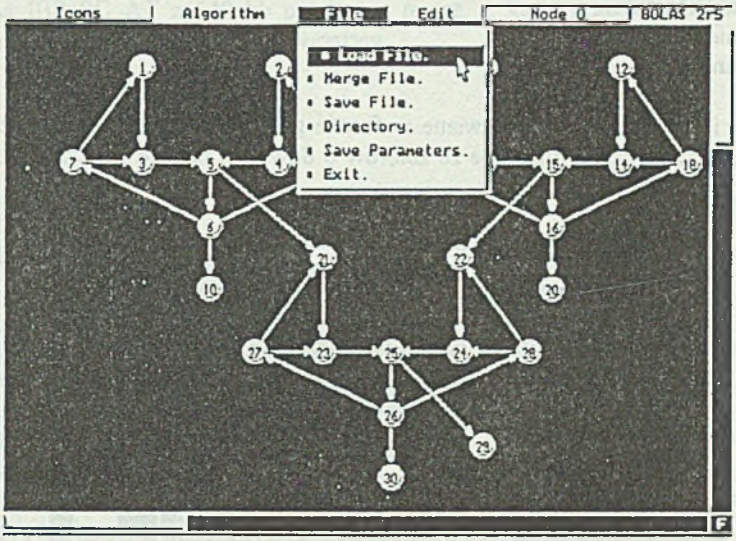
```
SUBROUTINE węzeł_9 (AND; Or; avoid : integer);
    begin
        if (ACTIVA<>1) then TOKEN(koniec, avoid, 'N')
      else TOKEN (nonsens, true, 'N');
    end;
```

{ Podprogram SUBROUTINE węzeł_9 wysyła komunikat do podprogramu o zmiennej wejściowej "koniec" i wartość "avoid" z wyjątkiem pierwszej swojej realizacji, podczas której wysyła komunikat do podprogramu o zmiennej wejściowej nonsens i wartość: true}

V. W celu zainicjowania symulacji programu równoległego program źródłowy musi zawierać wyrażenie żetonu "odpalającego" (init_token).

INIT_TOKEN (parametr, expression, type);
Parameter, expression i type ma takie samo znaczenie jak w wyrażeniu TOKEN.

Przykład:
init_TOKEN (avoid, 1, 'N');



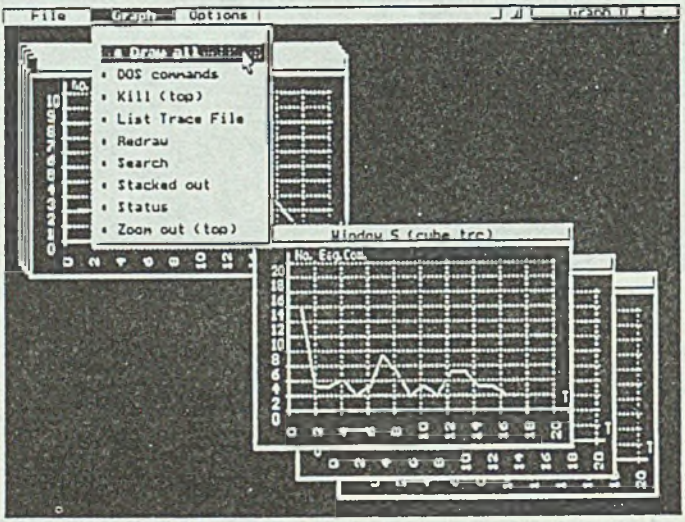
Rys.3. Symulator BOLAS – struktura grafowa algorytmu
Fig.3. BOLAS simulator – graph algorithm structure

5. Wizualizacja zachowania się systemu równoległego

Ze względu na złożoność systemów równoległych trudno jest znaleźć optymalne warunki pracy dla każdego algorytmu i architektury. By to osiągnąć, potrzebne jest narzędzie analizy i wizualizacji dużej ilości informacji o zachowaniu się systemu podczas realizacji algorytmu [2, 4, 5]. W PSEE znajdujemy takie narzędzie i uruchamiamy je poprzez uruchomienie programu graph.exe. GRAPH korzysta z danych zawartych w zbiorach o dowolnej nazwie i o rozszerzeniu .trc i stworzonych przez BOLAS podczas symulacji realizacji programu równoległego. Dokonywany podczas symulacji pomiar efektywności musi opierać się na zdefiniowanych i mierzalnych wielkościach takich, jak np.:

– węzły gotowe/czas - węzły gotowe do wykonania, lecz czekające na przydzielenie do procesora; pozwala nam to ocenić możliwości alternatywnego wyznaczenia kolejności wykonywania zadań lub ocenić potrzebę dodania dodatkowego procesora do systemu,
-- procesory zajęte/czas,
-- wolne procesory/czas,
-- początek komunikacji/czas,
-- koniec komunikacji/czas,
-- ilość komunikacji/czas,
-- przepełnienie łączy/czas,
-- diagram Gantta.

Powyższe informacje są przedstawiane w formie graficznej i użytkownik może tworzyć różne okienka zawierające informacje ze zbiorów o dowolnej nazwie i rozszerzeniach .trc z kilku różnych symulacji.



Rys.4. Blok GRAPH -- wizualizacja pomiarów
Fig.4. GRAPH block -- measurement vizualization

6. Zakończenie

Został opisany przyjazny dla użytkownika interfejs do oceny efektywności systemu równoległego. Dostarcza on szeregu informacji o zachowaniu się systemu równoległego podczas działania. Pozwala na rozwinięcie intuicji projektantowi systemów równoległych. Umożliwia nabycie doświadczenia przy projektowaniu i ocenianiu wpływu szeregu parametrów systemu równoległego na końcowy wskaźnik efektywności systemu. PSEE pracuje na IBP PC AT z 640 kb pamięci, z myszą, z kartą graficzną EGA/VGA i pod systemem operacyjnym DOS 3.3 lub wyższym.

LITERATURA

[1] Luque E., Ripoll A., Margalef T. i Hernandez P.: Static scheduling of parallel progam graphs including loops, Proceedings of Twenty-Sixth Annual Hawaii International Conference on System Science, IEEE Software, vol. 2, 1993, s. 526-535.
[2] Funka-Lea C., Kontogioros T., Morris R., Robin L.: Interactive visual modeling for performance, IEEE Softw., 1991, vol.9, s. 58-68.
[3] Luque E., Suppi R., Sorribes J.: Designning parallel systems: a performance prediction problem, Microprocessors and Microsystems, 1992, vol.16, no.1, s. 25-35.
[4] PSEE User's Manual, Dept. Informatica, Computer Architecture and Operating Systems Unit, Universitat Autonoma de Barcelona, November 1992.
[5] Luque E., Suppi R., Sorribes J., Mayosky M., Senar M.: Simulation and visualisation tools for link based parallel architecture, Microproces. Microprog, v.32, no.1-5, 1991, s.479-486
[6] Flynn M.: Same computer organizations and their effectiveness, IEEE Tran.Comp., vol.21, no.9, 1972, s. 948-960.
[7] Agrawal D., Janakiram V.: Evaluating the performance of multicomputer configuration, IEEE Comp., vol.19, no.5, 1986, s. 23-27.

Recenzent: Dr hab. inż. Stanisław Kozielski, Profesor Pol.Śląskiej

Wpłynęło do Redakcji 15 listopada 1994 r.

Abstract

PSSE is a software tool that provides a multiprocessor system for research into alternative architectural decisions and experimentation, with such issues as design, tuning, scheduling, clustering and routing policies. PSSE facilitaties simulation and performance evaluation as well as a prediction environment for the design and tuning of parallel systems. These tasks involve cycles through programming, simulation, measurement, visualisation and modification of parallel system parameters. Weighted Behavioural Directed Graph (WBG) has been selected as modelling formalizm whose components have been discribed in Table 1. PSSE includes a parallel programming tool, a simulator for link oriented parallel systems, BOLAS, and a performance evaluation tool, GRAPH. These PSSE modules are tools oriented to support the above tasks in user-friendly and interactive graphical form.