ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996

Seria: INFORMATYKA z. 31 Nr kol. 1348

Aleksandra WERNER

# GRAFICZNE PROGRAMOWANIE RÓWNOLEGŁE W SYSTEMIE HeNCE

Streszczenie. Niniejsze opracowanie prezentuje środowisko *HeNCE*, przeznaczone do wspomagania tworzenia programów równoległych uruchamianych w sieci stacji roboczych UNIX. Artykuł zawiera ponadto przykłady programów, napisanych przy użyciu graficznego narzędzia o nazwie *htool*, oraz analizę przeprowadzonych dla nich eksperymentów.

## THE GRAPHICAL PARALLEL PROGRAMING IN HeNCE SYSTEM

Summary. This paper presents environment *HeNCE*, designed to assist in developing parallel programs that run on a network of UNIX workstations. software environment called HeNCE. There are some examples (with their analyzing), written by the use of graphical tool: *htool*, in this article.

## PROGRAMATIN PARALELLE GRAPHIQUE DANS LE SYSTEMÉ HeNCE

Résumé. Cet article présent d'environnement *HeNCE*, destiné ŕ l'assistance pour création des programmes paralléles travaillant sur les ordinateurs, sous le systéme opérationnel UNIX, joignent en réseau local. Dans cet article sont aussi donnés les exemples et l'analyse des programmes écrites sous l'outil graphique „*htool*".

Pracę wykonano w ramach grantu nr KBN 3 P406 011 04.

# 1. Wstęp

HeNCE (Heterogeneous Network Computing Environment) jest przykładem środowiska realizującego graficzne podejście do programowania równoległego. Został opracowany przez naukowców z Carnegie Mellon University oraz Oak Ridge National Laboratory and University of Tennessee. HeNCE dostarcza użytkownikowi narzędzi umożliwiających łatwe i szybkie tworzenie (kompilowanie, śledzenie) oraz wykonywanie programów równoległych, działających na bazie systemu PVM.

PVM jest pakietem, pozwalającym wykorzystać niejednorodne sieci komputerowe (heterogeneous network) jako pojedynczy zasób obliczeniowy. PVM dostarcza narzędzi do komunikacji, synchronizacji oraz inicjacji procesów potomnych. Podczas gdy PVM dostarcza programiście narzędzi niskiego poziomu, do implementacji programów współbieżnych, HeNCE oferuje mu wyższy poziom abstrakcji do specyfikacji równoległości.

Głównym celem twórców HeNCE'a było stworzenie narzędzia, pozwalającego programiście na wykorzystanie kilku dostępnych mu komputerów, bez konieczności zgłębiania specyfiki programowania równoległego.

HeNCE oferuje programiście możliwość stworzenia programu współbieżnego na podstawie odpowiednio opisanego rysunku (grafu). Tworzony rysunek jest prostym grafem, w którym węzły reprezentują odwołania do konwencjonalnych sekwencyjnych procedur napisanych w językach C lub Fortran. Wektory reprezentują porządek wykonywania poszczególnych procedur. Procedura może zostać wykonana tylko wtedy, gdy wykonanie wszystkich poprzedzających ją procedur (węzłów) zostało zakończone.

# 2. Wymogi systemu

Dla poprawnego działania systemu HeNCE muszą być spełnione następujące warunki:

1. Stacje, będące składowymi maszyny wirtualnej, muszą pracować w systemie operacyjnym UNIX z X Windows (wersja X11R4 lub wyższa). Przykładowe stacje robocze, zapewniające niezbędne środki do pracy z programem ***htool***, to:

| | |
|---|---|
| Sun 4 (Sparc) | z systemem operacyjnym SunOS 4.1.3 |
| Sun 3 | z systemem operacyjnym SunOS 4.1.X |
| IBM RS/6000 | z systemem operacyjnym AIX 3.2.X |
| HP Precision Arch | z systemem operacyjnym HPUX. |
| DEC (Alpha) | z systemem operacyjnym OSF-1 |
| DEC (PMAX) | z systemem operacyjnym Ultrix. |

2. Zapewnienie dostępu do systemu PVM (zainstalowanego w katalogu $HOME użytkownika) w sieci, w której uruchamiane są programy napisane w HeNCE.

3. Ustawienie zmiennej środowiskowej PVM_ROOT tak, by wskazywała na podkatalog *pvm3*. (W pliku ".cshrc"- w przypadku korzystania ze standardowego interpretera systemu UnixC-shell- dopisanie polecenia: *setenv PVM_ROOT ~/pvm3*).

4. Uzupełnienie ścieżek poszukiwań o:
*<nazwa_kartoteki_z_syst_PVM>/lib*
oraz: *~/pvm3/bin/ARCH* (*ARCH* jest nazwą architektury komputera).

# 3. Instalacja programu

## 3.1. Instalacja PVM

Do istalacji HeNCE niezbędne jest wcześniejsze zainstalowanie PVM. Źródła systemu PVM znajdują się w sieci Internet po adresem:

*netlib2.cs.utk.edu*

w zbiorze *pvm3.3.7.tar.z.uu*

Instalacja polega na kompilacji źródeł, rozpakowanych w katalogu $HOME użytkownika, poleceniem ***make***, a następnie wykonaniu polecenia:

***mkdir $HOME/pvm3 $HOME/pvm3/bin***

## 3.2. Instalacja HeNCE

System HeNCE dostępny jest w sieci Internet na komputerze netlib.org w kartotece *hence* poprzez anonimowego ftp. Potrzebne zbiory zawarte są w jednym zarchiwizowanym i skompresowanym pliku o nazwie:

*hence-2.0-src.tar.Z.uu*

Przewodnik do instalacji programu HeNCE wraz z całą dokumentacją, znajduje się w zbiorze

*hence-2.0-doc.ps.Z.uu*

Instalacja polega na:
1. zdekompresowaniu i rozarchiwizowaniu pliku źródłowego poleceniami -odpowiednio: *uncompress* <nazwapliku> i: *tar -xvf* <nazwapliku>.
2. wykonaniu ciągu poleceń:

- ***xmkmf***
- ***make Makefiles***
- ***make clean*** - tylko w przypadku, gdy w komputerze była już zainstalowana wcześniejsza wersja HeNCE-a***make***
- ***make install***

W celu uproszczenia instalacji HeNCE-a twórcy systemu przygotowali alternatywne zbiory prekompilowane na poszczególne architektury komputerów. Są to zbiory o nazwie:

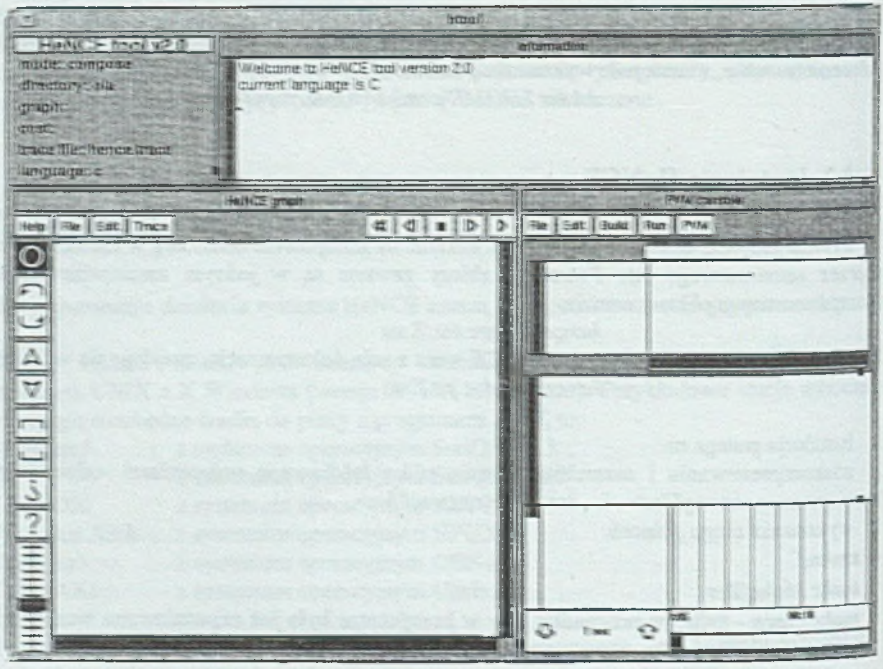
*hence-2.0-full.ARCH.tar.Z.uu*

(*ARCH* jest nazwą architektury komputera).

W skład systemu zarchiwizowanego w tych plikach wchodzą: program ***htool*** oraz prekompilowane biblioteki. Zbiory prekompilowane powinny być rozpakowane w kartotece głównej użytkownika. Zostaną one zapisane w kartotece *pvm3*. (Jeżeli -na przykład- użytkownik pracuje na komputerze SUN4 (Sun Spare running SunOs) i tam chce pisać i uruchamiać programy napisane w HeNCE, wtedy w podkartotece *pvm3* rozpakowuje i dekompresuje zbiór *hence-2.0-full.sun4.tar.Z.uu*).

# 4. Graficzne projektowanie struktur równoległych (htool)

## 4.1. Opis narzędzia



Rys. 1. Wygląd okna programu *htool*
Fig. 1. The *htool* window

*htool* jest graficznym narzędziem pozwalającym na proste i szybkie pisanie programów w HeNCE.

Po uruchomieniu *htool* na ekranie pojawia się pojedyncze okno podzielone na kilka części (rys.1). Każda z nich ma do spełnienia odrębne zadanie.

Segment zatytułowany "HeNCE tool v2.0" pozwala na wybór języka programowania ("C" lub "FORTRAN"), w którym programista będzie zapisywał procedury sekwencyjne wywoływane w danym węźle oraz na zmianę kartoteki, w której umiejscowiony został *htool* i w której będą zapamiętywane tworzone przez programistę zbiory.

Część o nazwie "Informations" ma za zadanie informować użytkownika o przebiegu kolejnych etapów powstawania programu (w tym o ewentualnych błędach popełnionych przez użytkownika podczas opisywania grafu).

Okno "HeNCE graph" służy głównie do projektowania grafu, ale również udostępnia ono użytkownikowi pasek narzędziowy do śledzenia wykonania programu napisanego w HeNCE.

W pozostałej części okna przedstawione są informacje dotyczące wirtualnej maszyny *PVM*, gdzie poza tablicą kosztów (czyli tablicą odzwierciedlającą przewidywany przez programistę koszt wykonania na danym komputerze danej procedury) oraz paskowym wykresem wykorzystania danej stacji, wyprowadzane są dane wyjściowe wykonywanego programu.

## 4.2. Etapy tworzenia programu równoległego

### 4.2.1. Rysowanie grafu

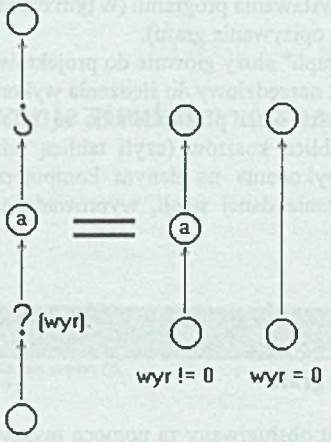
Program *htool* jest obsługiwany za pomocą myszy. Do dyspozycji użytkownika są ikony węzłów, za pomocą których może on utworzyć żądaną strukturę równoległą.

Istnieje kilka typów węzłów, mogących pojawić się w programie napisanym w środowisku HeNCE.

- Sekwencyjny węzeł obliczeniowy (**computation node**) jest podstawowym węzłem tworzonego grafu. Podczas wykonywania tego węzła wywoływany jest stosowny podprogram (w całości napisany przez programistę). Węzeł ten jest uaktywniany dopiero wtedy, gdy jego wszystkie poprzedniki (tzn. poprzedzające go węzły) zostały już uaktywnione.

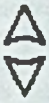
Pozostałe węzły są węzłami specjalnego przeznaczenia, gdyż definiują one sterowanie przepływem danych w grafie. Węzły te występują w parach, a pomiędzy węzłem początkowym (BEGIN) i końcowym (END) musi wystąpić podgraf. Węzły specjalnego przeznaczenia mogą być uważane za elementarne jednostki do modyfikacji grafu, ponieważ dodają one do bieżącego grafu podgrafy, na podstawie wyrażeń wyliczanych podczas wykonywania programu. Jakkolwiek konstrukcje te mogą być zagnieżdżane, nie jest dozwolone stosowanie ich przeplotu.

Para węzła warunkowego (*conditional node*) odpowiada instrukcji "if-then" w języku C. Program początkowego węzła warunkowego musi zawierać wyrażenie. Jeżeli wartość wyrażenia stanie się TRUE, wówczas podgraf zawarty pomiędzy węzłami BEGIN - END zostanie wykonany (podgraf będzie dodany do grafu). W przeciwnym wypadku podgraf nie jest wykonywany (jest pomijany, tzn. jest zastępowany przez wektor), a dalsze przetwarzanie programu rozpoczyna się od wierzchołka grafu, następującego po węźle END. (**Uwaga!** W systemie HENCE nie ma konstrukcji odpowiadającej strukturze "if-then-else").).



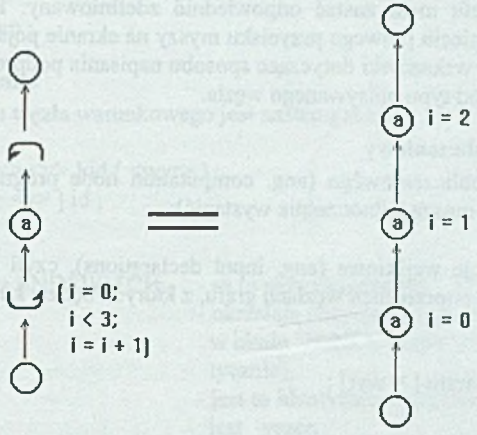

Rys. 2. Rozwinięcie podgrafu węzła warunkowego
Fig. 2. Conditional expansion

Para węzła pętli (*loop node)* definiuje iteracje (podobnie jak pętla "for" w języku "C"). Podgraf między węzłem początku i końca pętli jest powtarzany tyle razy, ile razy zmienna indeksowa pętli przyjmuje swoje kolejne wartości. Podgraf jest zastępowany przez sekwencję swoich kopii (rys. 3).

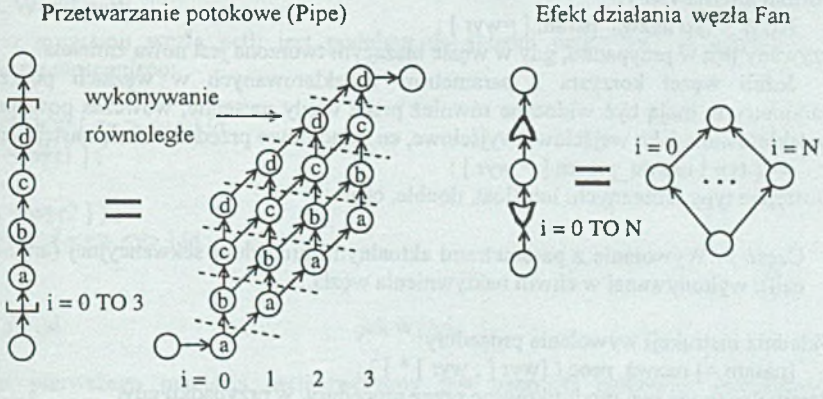


Przy współudziale pary węzła *fan node* tworzone są struktury równoległe. Podgraf ujęty węzłami "fan-out" (węzeł BEGIN) i "fan-in" (węzeł END) jest powtarzany, a kolejne jego repliki mają równolegle wyznaczane wartości. Ilość tworzonych węzłów jest sprecyzowana w programie węzła Fan Begin i może zależeć od zmiennych, których wartości obliczane sa dynamicznie podczas pracy programu.

Para węzła potokowego (*pipe node*) tworzy strukturę potoku. Rys. 4 przedstawia efekt potokowania (porównując go zarazem do działania węzła *fan*) i nakreśla, które węzły mogą być wykonywane równolegle.




Rys. 3. Rozwinięcie podgrafu węzła *Loop*
Fig. 3. *Loop* expansion




Rys. 4. Rozwinięcie podgrafu węzłów: *pipe* i *fan*
Fig. 4. Effect of *fan* and p*ipe*

### 4.2.2. Opisywanie grafu - podprogramy węzłowe

Każdy węzeł grafu musi zostać odpowiednio zdefiniowany. Po wskazaniu kursorem danego węzła i naciśnięciu prawego przycisku myszy na ekranie pojawia się okno edytora *vi*. Otwarty plik zawiera wskazówki dotyczące sposobu napisania podprogramu. Syntaktyka tego podprogramu zależy od typu opisywanego węzła.

#### 4.2.2.1. Węzeł obliczeniowy

Program węzła obliczeniowego (ang. computation node program) składa się z trzech części (nie wszystkie muszą jednocześnie wystąpić):

- Część 1. Deklaracje wejściowe (ang. input declarations), czyli specyfikacja zmiennych zdefiniowanych w poprzednich węzłach grafu, z których będzie korzystał węzeł bieżący.

Format deklaracji:

```
< [ typ ] nazwa_param [ = wyr] ;
```

gdzie:

| | |
|---|---|
| typ | - typ parametru |
| nazwa_param | - nazwa parametru |
| wyr | - początkowa wartość parametru |

Format alternatywny:

```
NEW < typ nazwa_param [ =wyr ] ;
```

używany jest w przypadku, gdy w węźle bieżącym tworzona jest nowa zmienna.

Jeżeli węzeł korzysta z parametrów zadeklarowanych w węzłach poprzednich, a parametry te mają być widoczne również przez węzły następne, wówczas powinny być one zadeklarowane jako wejściowe/wyjściowe, co składniowo przedstawia się następująco:

```
<> [ typ ] nazwa_param [ = wyr ] ;
```

Dostępne typy zmiennych: int, float, double, char

- Część 2. Wywołanie z parametrami aktualnymi procedury sekwencyjnej (ang. subroutine call), wykonywanej w chwili uaktywnienia węzła.

Składnia instrukcji wywołania procedury:

```
[param =] nazwa_proc ( [wyr [ , wyr ] * ] ) ;
```

Parametry mogą być modyfikowane przez procedury, w przypadku gdy:

a) następuje podstawienie pod parametr *param* wartości zwracanej przez procedurę (parametr ten nie musi być zadeklarowany):

```
np.          a = aktualizuj( );
```

b) jako argument procedury przesłane są adresy tych parametrów:

```
np.          aktualizuj( &a );
```

c) argumentem procedury jest tablica:

```
np.      aktualizuj( a[ ] );
```

- Część 3. Deklaracje wyjścia (ang. output declarations) - czyli definicje nowych zmiennych, które potem będą udostępnione następnym węzłom grafu.

Format deklaracji:

> <typ> nazwa_param ;

#### 4.2.2.2. Węzeł warunku

Syntaktyka programu węzła warunkowego jest następująca:

```
BEGINSWITCH [ <x> <y> ] id ( <wyr> ) ;
ENDSWITCH [ <x > <y> ] id ;
```

przy czym:

| | |
|---|---|
| BEGINSWITCH oraz ENDSWITCH - | - są to słowa kluczowe; |
| <x > <y > | - określają pozycję danego węzła warunkowego w oknie "HeNCE graph" (wstawiane automatycznie); |
| id | - jest to identyfikator etykiety, którą oznaczony jest węzeł; |
| wyr | - oznacza wyrażenie, na podstawie którego subgraf, zawarty między parą węzła warunku, będzie wykonywany (dołączony) lub też ignorowany. |

#### 4.2.2.3. Węzeł pętli

Składnia programu węzła pętli jest podobna do składni pętli "for" w języku C i przedstawia się następująco:

```
BEGINLOOP [ <x > <y >] id
( [ zm = wyr1 ] ;
    wyr ;
  [ zm = wyr2 ] ) ;
ENDLOOP [ <x > <y> ] id ;
```

przy czym:

| | |
|---|---|
| <x >, <y >,id | - jak wyżej. |

Podczas pierwszego przejścia pętli tworzony jest parametr całkowity (oznaczony identyfikatorem zm), a jego wartość jest określona przez wyrażenie wyr1. Jeżeli wartość środkowego wyrażenia jest różna od zera, wówczas podgraf wewnątrz pętli jest wykonywany. Następnie zmienna var jest aktualizowana za pomocą wyrażenia wyr2 i ponownie odbywa się sprawdzenie środkowego wyrażenia (warunku). Jeżeli jego wartość jest równa zero, wtedy wykonywanie programu rozpoczyna się od instrukcji następującej po słowie kluczowym 'ENDLOOP'.

#### 4.2.2.4. Węzeł potoku

Instrukcje programu węzła potokowego mają budowę analogiczną do instrukcji innych węzłów specjalnego przeznaczenia, a ich zapis jest następujący:

```
BEGINPIPE [ <x> <y> ] id
 zm = wyr1 TO wyr2 ;
ENDPIPE [ <x> <y> ] id ;
```

identyfikatory:

| | |
|---|---|
| <x > <y > | - określają pozycję danego węzła warunkowego w oknie "HeNCE graph" (wstawiane automatycznie); |
| id | - jest to identyfikator etykiety, którą oznaczony jest węzeł. |

Dla każdej wartości parametru zm tworzona jest odrębna kopia podgrafu zawartego między parą węzła potokowego. Równocześnie, dla każdego węzła podgrafu istnieje połączenie między jego egzemplarzem w jednej kopii podgrafu a jego egzemplarzem w następnej kopii podgrafu.

#### 4.2.2.5. Węzeł fanout/fanin

Budowa programu węzła Begin Fan przedstawia się następująco:

```
FANOUT [ <x> <y> ] id ;
 zm = wyr1 TO wyr2 ;
```

natomiast węzła End Fan:

```
FANIN [ <x> <y> ] id ;
```

gdzie:

| | |
|---|---|
| <x>, <y>, id | - jak wyżej. |

Wyrażenia wyr1 oraz wyr2 określają zakres wartości zmiennej zm, traktowanej jako parametr wyjściowy typu całkowitego. Podobnie jak dla węzła potokowego, dla każdej wartości zmiennej zm tworzona jest odrębna kopia podgrafu. Poszczególne kopie podgrafu mogą współpracować ze zmienną zm, traktując przypisaną jej wartość jako parametr całkowity.

### 4.2.3. Definicja procedur

Każdy wstawiony przez programistę węzeł obliczeniowy związany jest z procedurą sekwencyjną, której treść musi zdefiniować sam użytkownik. Może to zrobić na dwa sposoby:

1. Korzystając z programu ***htool***, otworzyć okno edytora vi (przez naciśnięcie kombinacji klawiszy: *Shift + prawy przycisk myszy*).

2. Zewnętrznym edytorem utworzyć plik o nazwie: *<nazwa_procedury>.c* , zawierający treść procedury danego węzła.

#### 4.2.4. Definiowanie tablicy kosztów

Tablica kosztów, definiowana przez użytkownika przed wykonaniem programu, pełni podwójną rolę:

1.definiuje listę komputerów wchodzących w skład równoległej maszyny wirtualnej,

2.dostarcza systemowi informacji o przewidywanych kosztach wykonania każdej procedury uruchamianego programu, na każdej z maszyn połączonych w maszynę wirtualną.

Jeżeli tablica kosztów nie zostanie przez programistę wypełniona, wówczas do wykonania programu zostanie użyta domyślna tablica, powodująca wykonanie wszystkich węzłów (procedur) w komputerze głównym (tzn. w stacji, na której pracuje *htool*). Oznacza to jednak, że dostępny jest tylko jeden procesor i program zostanie wykonany sekwencyjnie. Może to być pomocne jedynie przy uruchamianiu lub testowaniu programu.

Tablica kosztów jest tablicą, której wiersze są opisane identyfikatorami nazw komputerów składających się na komputer wirtualny, a kolumny - nazwami procedur składających się na wykonywany program. Programista wpisuje w każdym przecięciu wiersza i kolumny liczbę całkowitą, która odzwierciedla szacunkowy koszt wykonania danej procedury na danej stacji roboczej.

Tabela 1 przedstawia przykładową tablicę kosztów dla programu zawierającego trzy procedury: *przetwarzanie*, *wejście* oraz *wyjście*. Program będzie rozpraszany między dwa komputery o nazwach: *stacja1* i *stacja2*.

Tabela 1

Przykładowa tablica kosztów

| Nazwa stacji | Procedura | | |
|---|---|---|---|
| | przetwarzanie | wejście | wyjście |
| stacja1 | 4 | 2 | 6 |
| stacja2 | 1 | 3 | 2 |

Przykładowa tablica kosztów pokazuje, że wykonanie procedury *wyjście* na komputerze *stacja1* jest trzy razy "droższe" niż na komputerze *stacja2*, a procedury *przetwarzanie* - cztery razy droższe. Z kolei koszt wykonania procedury *wejście* na tej samej stacji jest tańszy. HeNCE będzie korzystał z tych informacji, próbując wykonać procedurę jak najtańszym kosztem.

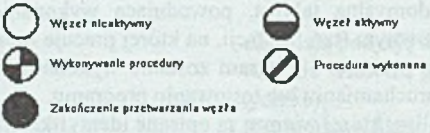
Jeżeli program napisany w HeNCE jest uruchamiany w sieci maszyn tego samego typu i trudno jest przewidzieć koszty wykonania poszczególnych procedur w danych komputerach, wówczas najefektywniej jest wypełnić całą tablicę taką samą wartością całkowitą. HeNCE będzie próbował tak rozesłać węzły, by uzyskać jak najlepszy efekt zrównoleglania.

### 4.3. Uruchamianie programu i jego śledzenie

Po narysowaniu i opisaniu grafu oraz wypełnieniu tablicy kosztów program musi zostać skompilowany. Dokonuje się tego przez wybranie z menu okna PVM Console opcji *Build*. Jeżeli kompilacja przebiegła poprawnie (ewentualne błędy pojawią się w oknie Informations), program może zostać wykonany. Wykonanie programu jest równie proste co jego skompilowanie, gdyż wymaga od programisty jedynie wyboru opcji *Run Program* z menu

*Run* okna PVM. Podczas wykonywania programu jego przebieg jest uwidoczniony w oknie HeNCE Graph poprzez zmiany wyglądu ikon grafu. Znaczenie tych zmian pokazane jest na rys.5. Zmianom kolorystycznym ulega także wypełniona wcześniej tablica kosztów (tabela 2).

Po zakończeniu wykonywania programu i pojawieniu się w segmencie PVM Console danych wyjściowych należy wyjść z trybu wykonywania (opcja *Exit Run Mode*), gdyż w przeciwnym przypadku niemożliwa będzie np. edycja grafu.




Rys. 5. Znaczenie zmian w wyglądzie węzłów grafu podczas wykonywania programu
Fig. 5. Meaning of runtime icons

Tabela 2

Znaczenie kolorów komórek tablicy kosztów

| Monitor kolorowy | Monitor monochromatyczny | Znaczenie zmian |
|---|---|---|
| Zielony | Ciemnoszary | Wykonywanie węzła |
| Żółty | Jasnoszary | Węzeł niewykonywany |
| Szary | Biały | Brak zadań |

Dokładniejsze przeanalizowanie etapów wykonywania poszczególnych procedur jest możliwe w trybie Trace Mode, do którego przejście odbywa się przez wybór opcji *Load Trace File* (okno HeNCE Graph). Domyślnie, zbiorem zawierającym ślad programu jest *hence.tr*. Informacje nagrane w tym zbiorze mogą być odtworzone za pomocą przycisków, służących kolejno do: cofania się (praca ciągła oraz krok po kroku), zatrzymania śledzenia, śledzenia wykonania programu od początku (analizowanie krok po kroku oraz ciągłe) (rys.6).

Rys. 6. Przyciski służące do śledzenia programu
Fig. 6. Trace replay control button

System HeNCE oferuje możliwość krokowego śledzenia wykonania programu ze zmienną, kontrolowaną prędkością w granicach od 100 razy wolniej do 100 razy szybciej od rzeczywistej szybkości wykonania programu.

Przed wyjściem z programu ***htool*** programista obowiązany jest do zatrzymania PVM (opcja *Halt PVM*).

# 5. Przykłady

## 5.1. Całkowanie funkcji w przedziale [a, b]

Funkcją całkowaną jest f(x) = 1/x w przedziale: [3, 10]. Przedział ten jest dzielony na dwie połowy, w których - równolegle - odbywa się całkowanie. Uzyskane wyniki są następnie sumowane.

W skład programu wchodzą cztery funkcje:

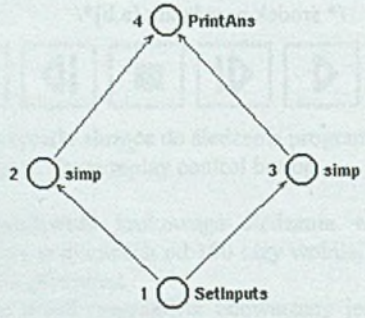
- void **SetInputs**(a, b, mid, n)
  /* Określanie wartości danych wejściowych programu i wyliczanie punktu środkowego przedziału, w którym rozpatrujemy funkcję f(x).*/
- static double **f**(x)
  /* f jest funkcją całkowaną.*/
- double **simp**(a, b, n)
  /* Całkowanie funkcji w przedziale [a, b] */
- void **PrintAns**(s1, s2)
  /* Sumowanie wyników cząstkowych (s1 oraz s2) i wydruk wyniku na standardowe wyjście.*/

Poniżej zamieszczone są definicje funkcji:

```
void SetInputs(a, b, mid, n)
  double *a;
  double *b;
  double *mid;
  int *n;
{
  *a = 3.0;                /* początek przedziału */
  *b = 10.0;               /* koniec przedziału */
  *n = 100;                /* liczba punktów w każdym z podprzedziałów */

  *mid = (*a + *b)/2.0;    /* środek przedziału [a,b]*/
}

static double f(x)
{
  double x;
return 1.0/x;
}

double simp(a, b, n)
  double a;
  double b;
  int n;
```

```
{
  double h, half, hov2, s, x;
  int nm1, i;

  if (n < 3) n = 3;
  h = (b - a)/n;              /* długość badanych podprzedziałów */
  hov2 = h/2.0;
  s = 0.0;                    /* wartość całki funkcji f(x) w przedziale [a, b] */
  half = f(a + hov2);
  nm1 = n - 1;

  for (i = 1; i <= nm1; i++) {
    x = a+ i*h;
    s = s + f(x);
    half = half + f(x + hov2);
  }
  s = (h/6.0)*(f(a) + 4.0*half + 2.0*s + f(b));
  return s;
}
#include <stdio.h>
void PrintAns(s1, s2)
  double s1;
  double s2;
{
  printf("wynik po integracji funkcji: %f\n", s1+s2);
  fflush(stdout);
  sleep(5);
}
```

W systemie HeNCE program całkujący funkcję f(x) = 1/x opisany jest następującym grafem:

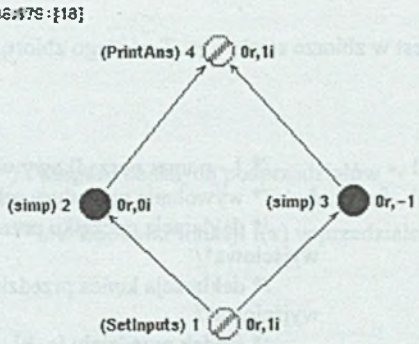



Rys.7. Graf programu całkującego
Fig.7. Graph of integrating program

Graf ten zapisany jest w zbiorze *static_i.gr*. Treść tego zbioru jest następująca:

```
PROGRAM

NODE [ 202 335 ] 1                  /* 1 - numer porządkowy węzła*/
  SetInputs(&a, &b, &mid, &n); /* wywołanie procedury sekwencyjnej*/
  > double a;                       /* deklaracja początku przedziału całkowania - zmienna
                                    wyjściowa*/
  > double b;                       /* deklaracja końca przedziału całkowania - zmienna
                                    wyjściowa*/
  > double mid;                     /* środek przedziału [a, b] - zmienna wyjściowa*/
  > int n;                          /* liczba punktów w każdym z podprzedziałów*/

NODE [ 118 257 ] 2                  /* 2 - numer porządkowy węzła*/
  < double a;
  < double mid;
  < int n;
  s1 = simp(a, mid, n);
  > double s1;                      /* całka funkcji f(x) w przedziale [a, mid] - zmienna
                                       wyjściowa*/

NODE [ 286 255 ] 3
  < double b;
  < double mid;
  < int n;
  s2 = simp(mid, b, n);
  > double s2;                      /* całka funkcji f(x) w przedziale [mid, b] - zmienna
                                       wyjściowa*/

NODE [ 198 157 ] 4
  < double s1;
  < double s2;
   PrintAns(s1, s2);
/* opis połączeń między węzłami*/
ARC 1 2                             /* połączenie między węzłem o numerze porządkowym
                                    1 a węzłem o numerze 2*/

ARC 1 3
ARC 2 4
ARC 3 4
```

Po wykonaniu programu graf zmienia się w sposób przedstawiony na rys. 8:




Rys.8. Zmiana wyglądu grafu po wykonaniu programu
Fig.8. Graph's changes after executing

Dodatkowo, obok grafu, pojawia się czas wykonania programu (w tym wypadku: 6.179 s).

### 5.1.1. Przykładowe czasy wykonania zadania

Tabela 3

Uzyskane czasy wykonania zadania opisanego w punkcie 5.1

| Lp. | Tablica kosztów | | | | Czas wykonania zadania |
|---|---|---|---|---|---|
| | konfiguracja maszyny wirtualnej | szacunkowy koszt wykonania procedury | | | |
| | | SetInputs | simp | PrintAns | |
| 1. | SPARCclassic | 3 | 2 | 2 | 6.179 sek |
| 2. | SPARCclassic | 4 | 5 | 2 | |
| | SPARCstation IPX | 3 | 4 | 3 | 5.952 sek |
| 3. | SPARCclassic | 9 | 8 | 10 | |
| | SPARCstation10 | 6 | 4 | 7 | 13.772 sek |
| | SPARCstation IPX | 7 | 1 | 3 | |
| 4. | SPARCclassic | 1 | 6 | 7 | |
| | SPARCclassic | 3 | 4 | 8 | 6.153 sek |
| | SPARCstation IPX | 2 | 1 | 3 | |
| | SPARCstation SLC | 4 | 1 | 10 | |

### 5.1.2. Analiza wyników

Analiza czasów wykonania zadania wskazuje na to, iż bardzo istotny dla uzyskanych wyników jest sposób wypełnienia tablicy kosztów. Przy jej uzupełnianiu należy wziąć pod uwagę nie tylko moc obliczeniową danej stacji roboczej, lecz również stopień jej obciążenia.

W przypadku błędnego oszacowania powyższych parametrów stacji uzyskuje się znacznie dłuższe czasy wykonania zadania (wynik: 13.772 sek w wierszu 3.).

## 5.2. Iloczyn skalarny dwóch wektorów

Iloczyn skalarny dwóch wektorów jest sumą iloczynów ich składowych. W programie obliczany jest iloczyn skalarny wektorów: *wekta* oraz *wektb*; ilość składowych tych wektorów równa się 100.

W kolejnych podpunktach rozdziału przedstawione zostały różne wersje rozwiązania powyższego zadania.

### 5.2.1. Wersja sekwencyjna (napisana w języku C)

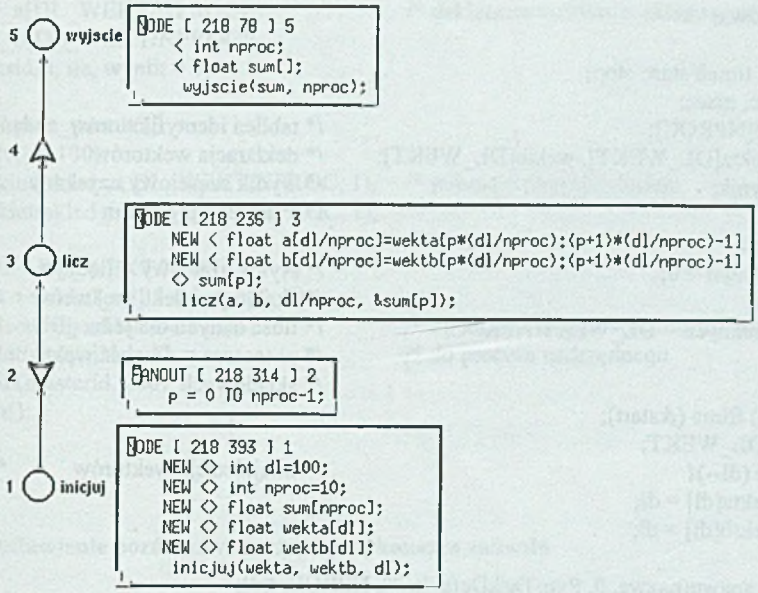
```
#include <sys/time.h>
#include <sys/timeb.h>
#include <stdio.h>
#define DL_WEKT 100                          /* długość wektorów  */
main()
{
    int wekta[DL_WEKT], wektb[DL_WEKT];      /* deklaracja wektorów       */
    int i, dl = DL_WEKT;
    int iloczyn = 0;                         /* wynik =il. skalarny wekt. */
    int sec, msec;
    struct timeb start, stop;

    (void) ftime (&start);
    while (dl--){                            /* inicjalizacja wektorów    */
      wekta[dl] = dl;
      wektb[dl] = dl;
    }
    dl = DL_WEKT;
    for (i=0; i<dl; i++) {
      iloczyn += wekta[i] * wektb[i];
    }
    printf ("Iloczyn skalarny = %d\n", iloczyn);
    (void) ftime(&stop);
    sec = (int) (stop.time - start.time);
    msec = (int) (stop.millitm - start.millitm);
    if (msec < 0) {
      sec -= 1;
      msec += 1000;
    }
    printf("czas pracy %d sec %d ms\n", sec, msec);
    exit(0);
}
```

### 5.2.2. Rozwiązanie w systemie HeNCE

Wektory dzielone są na podwektory. Iloczyn skalarny podwektorów liczony jest równolegle (wykorzystano węzeł *fan*). Graf programu ma postać jak na rys.9, a obok poszczególnych węzłów zamieszczono odpowiadające im programy węzłowe.

Funkcje, wywoływane w poszczególnych węzłach, mają następującą postać:

```
/*
 * Inicjalizacja wektorów
*/
void inicjuj(wekta, wektb, dl)
  float *wekta, *wektb;                    /* deklaracja wektorów                  */
  int dl;                                  /* długość wektorów */
{
  while (dl--) {
    wekta[dl] = dl;                        /* przypisanie poszczeg. elementom  */
    wektb[dl] = dl;                        /* wektorów wartości początkowych */
  }
}
/*
 * iloczyn skalarny podwektorów: a i b.
*/
void licz(a, b, dl, s)
  float *a, *b;                            /* wektory: a oraz b                    */
  int dl;                                  /* długość a i b                        */
  float *s;                                /* wynik:  il. skalarny wektorów: a i b */
{
  *s = 0;
  while (dl--) {
    *s += a[dl]*b[dl];
  }
}
/*
 * Sumowanie iloczynów skalarnych podwektorów i wydruk wyniku
*/
void wyjscie(s, dl)
  float *s;                                /* wektor iloczynów skalarnych podwektorów */
  int dl;                                  /* długość wektora s                    */
{
  float total = 0.0;
  while (dl--) {
    total += s[dl];
  }
  printf("iloczyn skalarny wektorów wekta i wektb, wynosi:%f\n", total);
  fflush(stdout);
  sleep(5);
}
```




Rys.9. Graf programu obliczającego iloczyn skalarny dwóch wektorów
Fig.9. HeNCE graph for dot product program

### 5.2.3. Rozwiązanie przy użyciu systemu PVM

```
#include <sys/time.h>
#include <sys/timeb.h>
#include <stdio.h>
#include "pvm3.h"

#define DL_WEKT 100                    /* długość wektorów        */
#define NPROC   5                      /* ilość procesorów        */

main(argc, argv)
int argc;
char **argv;
{
  if (pvm_parent() == PvmNoParent)
    glowny (argv[0]);
  else
    potomny ();
}
```

```
glowny (nazwa)

char *nazwa;
{
   struct timeb start, stop;
   int sec, msec;
   int tid[NPROC];                          /* tablica identyfikatorów  zadań */
   int wekta[DL_WEKT], wektb[DL_WEKT];      /* deklaracja wektorów            */
   int wynik;                               /* wynik częściowy uzyskany       */
                                            /* z jednego procesora            */
   int i, dl;
   int iloczyn = 0;                         /* wynik końcowy = iloczyn        */
                                            /* skalarny zadekl. wektorów      */
   int iledanych = DL_WEKT/NPROC;           /* ilość danych dla jednego       */
                                            /* procesora = długość wekt.      */
                                            /* składowych                     */
   (void) ftime (&start);
   dl = DL_WEKT;
   while (dl--){                            /* inicjalizacja wektorów         */
      wekta[dl] = dl;
      wektb[dl] = dl;
   }
   pvm_spawn(nazwa, 0, PvmTaskDefault, "", NPROC, tid);
   for (i=0; i<NPROC; i++) {                /* przesłanie danych pocz.        */
      pvm_initsend (PvmDataDefault);        /* do procesów potomnych          */
      pvm_pkint (&wekta[i*iledanych], iledanych, 1);
      pvm_pkint (&wektb[i*iledanych], iledanych, 1);
      pvm_send (tid[i], 100);
   }
   for (i=0; i<NPROC; i++) {                /* odczyt danych z poszczeg.      */
      pvm_recv(-1, 200);                    /* procesów                       */
      pvm_upkint (&wynik, 1, 1);
      iloczyn += wynik;
   }
   printf ("Iloczyn skalarny = %d\n", iloczyn);
   (void) ftime(&stop);
   sec = (int) (stop.time - start.time);
   msec = (int) (stop.millitm - start.millitm);
   if (msec < 0) {
      sec -= 1;
      msec += 1000;
   }
   printf("czas pracy %d sec %d ms\n", sec, msec);
   pvm_exit();                              /* opuszczenie systemu PVM        */
   exit(0);
}
```

```
potomny()
{
  int sklad_a[DL_WEKT/NPROC];              /* deklaracja wektorów składowych */
  int sklad_b[DL_WEKT/NPROC];
  int masterid, i, ile, wynik = 0;

  masterid=pvm_parent();
  pvm_recv(-1, 100);
  pvm_upkint(sklad_a, DL_WEKT/NPROC, 1);   /* pobranie danych z bufora       */
  pvm_upkint(sklad_b, DL_WEKT/NPROC, 1);

  for (i=0; i < DL_WEKT/NPROC; i++)
    wynik += sklad_a[i] * sklad_b[i];
  pvm_initsend(PvmDataDefault);            /* zwrócenie częściowych wyników */
  pvm_pkint(&wynik, 1, 1);                 /* do procesu nadrzędnego        */
  pvm_send(masterid, 200);
  pvm_exit();
  exit(0);
}
```

### 5.2.4. Zestawienie porównawcze czasów wykonania zadania

Tabela 4

Porównanie szybkości działania programu w wersji sekwencyjnej (C) i równoległej

| Konfiguracja maszyny wirtualnej | Czas wykonania zadania obliczania iloczynu skalarnego wektorów o długości 10000 | | |
|---|---|---|---|
| | wersja sekwencyjna | rozwiązanie w HeNCE [1] | rozwiązanie w PVM |
| SPARCstation10 | 89.489 sek | 7.093 sek | 1.213 sek |
| SPARCstation10<br>SPARCstation IPX | | 7.132 sek | 0.602 sek |
| SPARCstation10<br>SPARCstation IPX<br>SPARCstation SLC | | 7.314 sek | 1.021 sek |
| SPARCstation10<br>SPARCstation IPX<br>SPARCstation SLC<br>SPARCclassic1<br>SPARCclassic2 | | 6.804 sek | 0.523 msek |

[1] Tablica kosztów została wypełniona samymi jedynkami.

### 5.2.5. Wnioski

Wartości zamieszczone w tabeli wykazują, pod względem szybkości działania, przewagę rozwiązania zadania w systemie PVM nad rozwiązaniami: sekwencyjnym, zapisanym w języku C, oraz w sytemie HeNCE (tabela 3). Jest to zrozumiałe ze względu na to, że HeNCE ma dodatkowe narzuty czasowe na prezentację graficzną kolejnych etapów wykonania programu. Przy zwiększaniu (zmniejszaniu) długości badanych wektorów narzuty te pozostają w przybliżeniu stałe. Bardzo duży wpływ na uzyskane wyniki ma również aktualne obciążenie stacji roboczych, będących elementami składowymi maszyny wirtualnej.

Przeprowadzono również dodatkowe eksperymenty badające wpływ długości wektorów na czas wykonania danego programu. Wykazały one, że dla wektorów o niewielkiej ilości składowych (np. 100 elementów) najszybciej wykonywany jest program sekwencyjny. Jest to spowodowane tym, że w systemie PVM dla tak małej liczby danych większość czasu tracona jest na komunikację i synchronizację procesów, a nie na operacje arytmetyczno-logiczne.

## 6. Wady i zalety systemu

Programy napisane w HeNCE są dzięki maszynie wirtualnej niezależne od platformy sprzętowej i systemu operacyjnego, na którym pracują.

Bardzo wygodne dla programisty jest korzystanie ze środowiska w postaci bibliotek do języków C i Fortran.

HeNCE nie wymaga od programisty konieczności uczenia się tradycyjnych elementarnych funkcji programowania równoległego.

W związku z tym, że program ***htool*** jest w dalszym ciągu rozwijany, występują w nim pewne ograniczenia:

- nie jest możliwe przypisanie wartości elementu tablicy do zmiennej (np. zm = tab[1]),
- brak ostrzeżenia w przypadku wyjścia z trybu edycji bez zachowania istniejących zmian,
- czasami system zachowuje się niestabilnie, co objawia się nieprawidłowym wyprowadzaniem danych wyjściowych (dane nie pojawiają się w przeznaczonym do tego celu oknie wyjściowym).

## LITERATURA

[1] Beguelin A., Dongara J.J., Geist G.A., Manchek R., Sunderam V.: HeNCE: A Users' Guide version 2.0. Carnegie Mellon University, 1995.

[2] Beguelin A., Dongara J., Geist G.A., Manchek R.: Graphical Development Tools for Network-Based Concurrent Supercomputing. Oak Ridge National Laboratory, 1995.

[3] Geist A., Beguelin A., Dongara J., Weicheng J., Manchek R.: PVM: Parallel Virtual Machine. Oak Ridge National Laboratory, 1994.

Recenzent: Dr inż. Ryszard Winiarczyk

Wpłynęło do Redakcji 18 marca 1996 r.

## Abstract

This article describes an X-Windows based environment *HeNCE* which relies on the PVM system, intended to assist users in parallel programming. In HeNCE, the programmer is responsible for specifying paralleism by drawing graph which express the dependiences and control flow of a program. There are six types of constructs in HeNCE graphs: subroutine nodes, dependency arcs, conditional (Fig. 2), loop (Fig. 3), fan (Fig. 4) and pipe constructs (Fig. 4). HeNCE programmer must additionally indicate which machines will be used to execute program. He does it, by specify an estimated cost for running subroutines on particular machines in special *cost matrix* (Table 1).

A graphical user interface *-htool-* is provided for writting HeNCE programs (Fig. 1). This tool provides an interface for creating HeNCE graph, configuring virtual machine, executing and analyzing HeNCE programs.

Because of the facts that nodes in the graph are subroutines written in either C or Fortran (e.g. procedures can be taken from existing code) and that environment provides facilities for visualizing trace information, HeNCE seemed to be very useful and easy software package developing parallel programing.