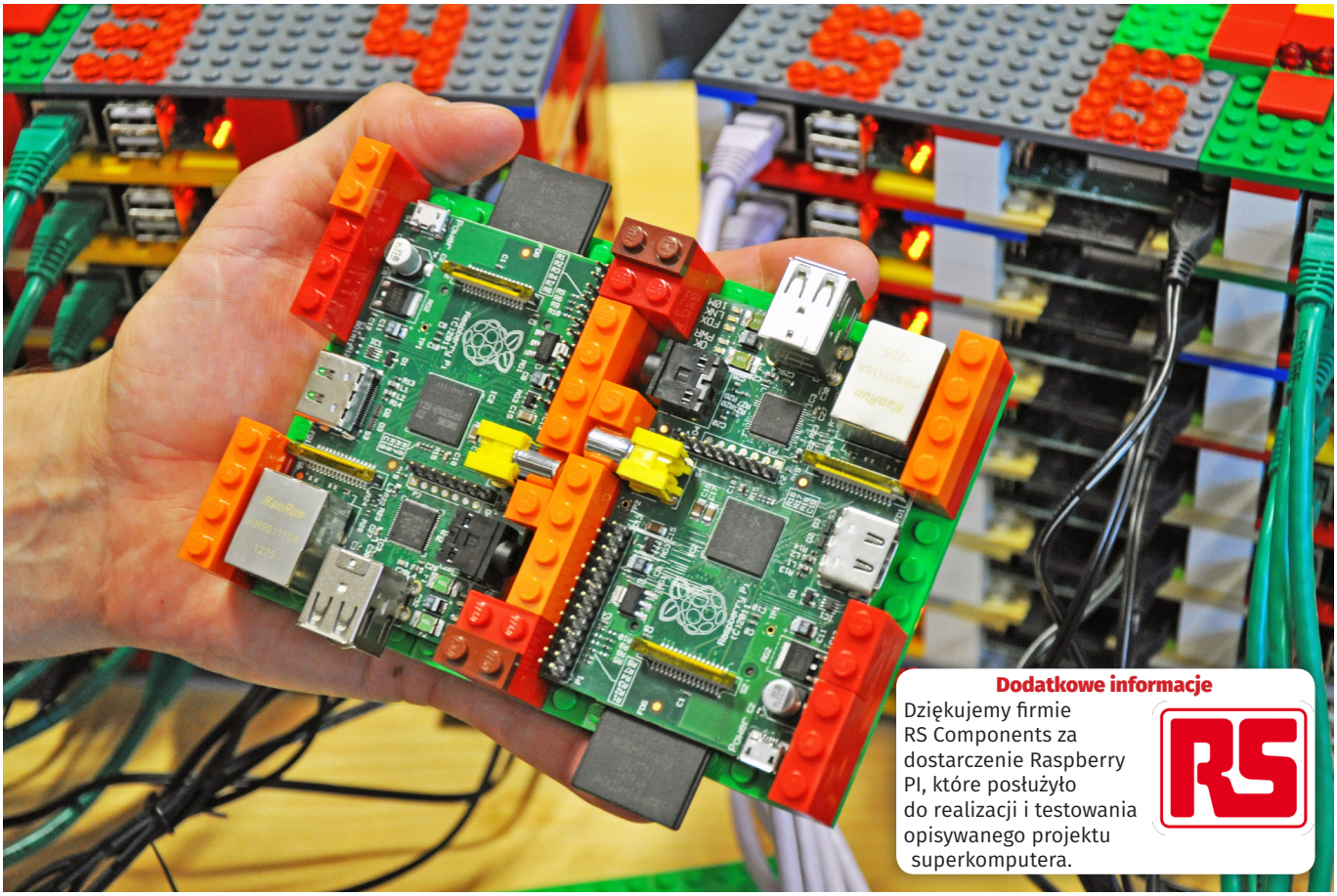


Dodatkowe informacje

Dziękujemy firmie RS Components za dostarczenie Raspberry PI, które posłużyło do realizacji i testowania opisywanego projektu superkomputera.

Fotografia 1. Raspberry PI obudowane klockami LEGO

Superkomputer z Raspberry PI

Moc obliczeniowa podstawowej wersji Raspberry PI nie jest duża – kształtuje się na poziomie wydajności procesora Intel Pentium II 300 MHz, a więc układu z końca ubiegłego wieku. Gdyby jednak połączyć ze sobą większą liczbę tych miniaturowych komputerów i w jakiś sposób dzielić wykonywane operacje pomiędzy poszczególne rdzenie, można by było uzyskać całkiem dużą sumaryczną wydajność. Tym bardziej jeśli skorzystać ze zintegrowanego GPU. W artykule pokazujemy jak zbudować skalowalny superkomputer, oparty właśnie o Raspberry PI. Co więcej, zastosowaną technikę da się zaimplementować na innych platformach komputerowych, a powstały system jest w stanie realizować programy napisane dla profesjonalnych superkomputerów.

Niska cena Raspberry PI sprawiła, że komputer ten jest wszechobecny. Znaleźć go można w wielu domach, często w wielu egzemplarzach. Dodatkowo, biorąc pod uwagę fakt, że nowsze wersje Raspberry PI są bardziej wydajne i z pewnością część użytkowników zastąpi dotychczasowe minikomputery nowszymi, liczba „leżących odłogiem" Raspberry PI z pewnością będzie szybko rosła.

Nieużywane Raspberry PI można w całkiem łatwy sposób wykorzystać do stworzenia systemu o cechach superkomputera, a więc maszyny przetwarzającej dane równolegle i zdolnej do wykonywania programów napisanych dla superkomputerów. Sumaryczna moc obliczeniowa, tak jak w przypadku wszystkich nowoczesnych superkomputerów, zależeć będzie przede wszystkim od liczby modułów, a więc liczby rdzeni przetwarzających dane i ich zegarów taktujących. W przypadku Raspberry PI wiele będzie zależało od zastosowanej generacji mikrokomputera.

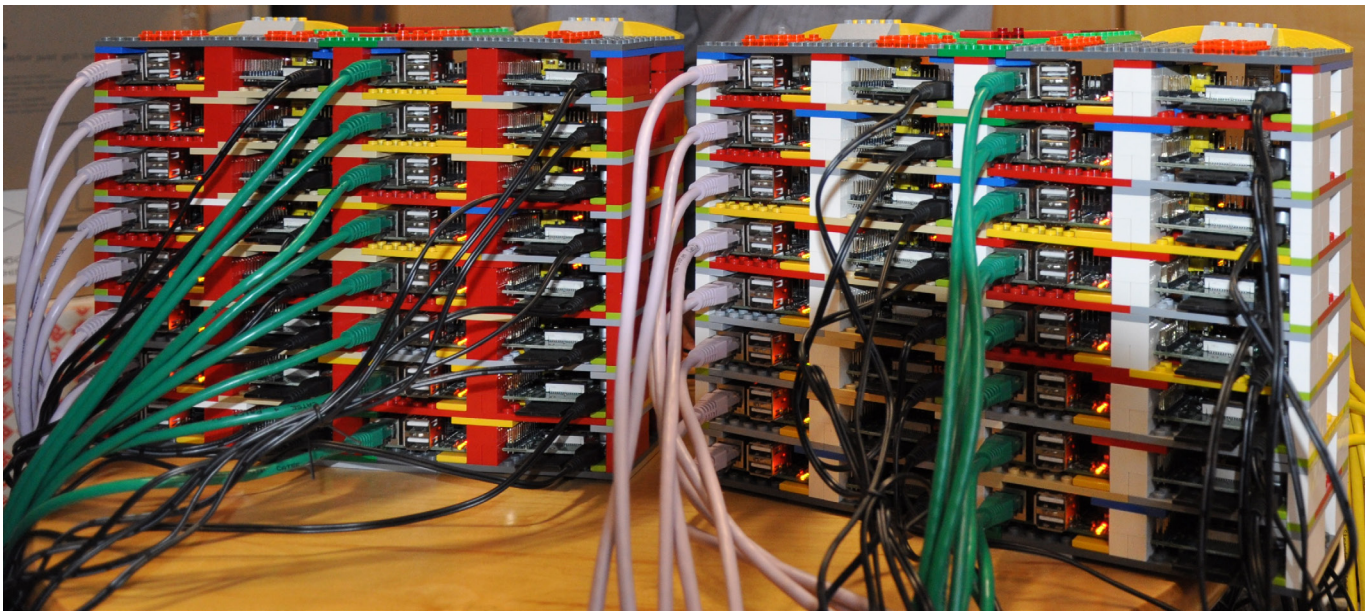
Obliczanie wydajności

Raspberry PI 1 zostało wyposażone w układ Broadcom BCM2835 z pojedynczym, 32-bitowym rdzeniem ARM1176JZF-S, taktowanym zegarem 700 MHz. Raspberry PI 2 ma już układ Broadcom BCM2836, a więc cztery rdzenie Cortex-A7, taktowane zegarem 900 MHz. Najnowsze Raspberry PI 3 poszło o krok dalej – zastosowany układ to 64-bitowy Broadcom BCM2837, z czterema rdzeniami Cortex-A53, taktowanymi zegarem 1,2 GHz. Można też wspomnieć o module Raspberry PI Zero, z układem takim jak w Raspberry PI 1, ale szybszym, bo taktowanym zegarem 1 GHz (**tabela 1**).

Moc obliczeniową komputera można przedstawiać na kilka sposobów. W przypadku typowych procesorów najczęściej porównanie opiera się na jednostce MIPS, tj. milionach wykonywanych instrukcji na sekundę (Mega Instructions Per Second). Nie trzeba być specjalistą od architektury

Tabela 1. Podstawowe parametry poszczególnych modeli Raspberry PI

Parametr\ Model	Raspberry PI 1 Model A i Model A+	Raspberry PI 1 Model B i Model B+	Raspberry PI 2 Model B	Raspberry PI 3 Model B	Raspberry PI Zero
SoC	Broadcom BMC2835		Broadcom BCM2836	Broadcom BCM2837	Broadcom BCM2835
CPU	1-rdzeniowy ARM1176JZF-S @700 MHz		4-rdzeniowy ARM Cortex-A7 @900 MHz	64-bitowy, 4-rdzeniowy ARM Cortex-A53 @1,2 GHz	1-rdzeniowy ARM1176JZF-S @1 GHz
GPU	Broadcom VideoCore IV, 24 GFLOPS			Broadcom VideoCore IV, 28,8 GFLOPS	Broadcom VideoCore IV, 24 GFLOPS
Pamięć RAM	256 MB	512 MB (starsze wersje mogą mieć 256 MB)	1 GB		512 MB



Fotografia 2. Superkomputer zbudowany przez Simona Coxa z Uniwersytetu Southampton. Na maszynę składają się 64 Raspberry PI, umieszczone na stelażu wykonanym z klocków LEGO.

komputerów, by wiedzieć, że różne procesory w ramach jednej instrukcji są w stanie wykonać różne liczby operacji, a tym bardziej – przetworzyć różne ilości danych. Liczba wykonywanych operacji zależy od złożoności poleceń kodu maszynowego i w efekcie dostępności nietypowych instrukcji, a ilość przetwarzanych danych w ramach jednej operacji w istotnym stopniu zależy też od liczby bitów danych, przyjmowanych przez instrukcję i wielkości rejestrów. Naturalnie, komputery 64-bitowe będą pod tym względem w uproszczeniu, 2 razy bardziej wydajne niż jednostki 32-bitowe. W efekcie wydajność podawaną w MIPSach nierzadko wylicza się w oparciu o praktyczne testy. Jednym z takich testów jest Dhrystone, który (ze względów historycznych) porównuje wydajność badanego komputera do minikomputera VAX 11/780 z 1977 roku, a którego wydajność w operacjach na liczbach całkowitych szacowana była na 1 MIPS.

Suche obliczenia mogłyby wskazywać, że 4 razy więcej rdzeni w Raspberry PI 2 względem RPI 1 oraz o niecałe 30% szybsze taktowanie powinno skutkować trochę ponad 5-krotnym wzrostem wydajności. Natomiast 64-bitowa architektura i zegar 1,2 GHz powinny sprawić, że Raspberry PI 3 będzie niemal 14 razy szybsze niż RPI 1. W praktyce jednak, w operacjach na liczbach całkowitych, testy wskazują jedynie ok. dwukrotny wzrost wydajności pomiędzy Raspberry PI 1 a Raspberry PI 2 i trochę ponad półtorakrotny pomiędzy Raspberry PI 2 a Raspberry PI 3. W efekcie Raspberry PI 3 w testach takich obliczeń jest niecałe 3 razy szybsze niż Raspberry PI 1, jeśli nie stosuje się podkręcania zegara. Raspberry PI 1 uzyskuje około 850 DMIPS (Dhrystone MIPS), Raspberry PI 2 ok 1540 DMIPS, a Raspberry PI 3 – 2440 DMIPS. Dla porównania, procesory Intel Core i7 uzyskują od ok. 15000 do 35000 DMIPS, w zależności od modelu, taktowania, dodatkowych zestawów instrukcji wykorzystywanych w trakcie testu oraz kompilatora i systemu operacyjnego. Można by się więc spodziewać, że 30 połączonych ze sobą Raspberry PI 1 (lub 10 Raspberry PI 3) ma szansę konkurować z nowoczesnymi (nieserwerowymi) jednostkami Intela z najwyższej półki.

A w jaki sposób taki klaster Raspberry PI można porównać do superkomputerów? Tu pojawia się inny problem – wydajność superkomputerów od lat liczona jest we FLOPSach, a więc w liczbie operacji zmiennoprzecinkowych, które maszyna jest w stanie wykonać w ciągu sekundy (Floating Point Operations Per Second). Co więcej, wydajność będzie różna, w zależności od tego, czy mowa o operacjach na liczbach pojedynczej czy podwójnej precyzji.

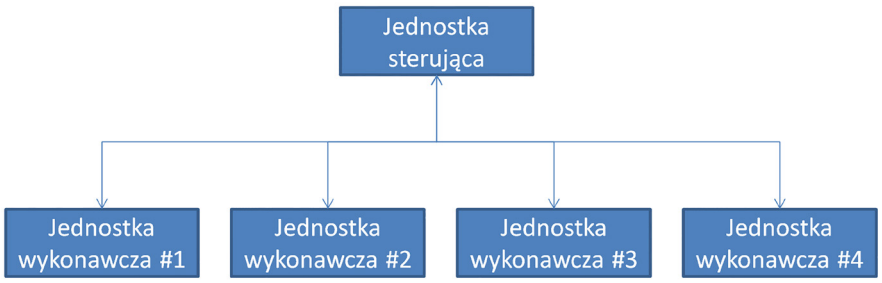
CPU pierwszego Raspberry PI ma wydajność rzędu 40 MFLOPS dla liczb podwójnej precyzji i 60 MFLOPS dla liczb pojedynczej precyzji. CPU drugiego Raspberry PI ma wydajność rzędu 120-150 MFLOPS dla liczb podwójnej precyzji (w zależności od testu) i ok. 155 MFLOPS dla pojedynczej precyzji. RPI 3 uzyskuje ok. 180 – 190 MFLOPS dla obu precyzji, ze względu na 64-bitowy rdzeń CPU.

Listing 1. Treść pliku machinefile, zawierającego listę jednostek wykonawczych. W przykładzie dostępne są trzy jednostki, przy czym trzecia z nich jest w stanie obsłużyć cztery wątki, a pozostałe po jednym wątku. Czwarta jednostka jest tymczasowo wyłączona znakiem komentarza

```
192.168.0.11
192.168.0.12:1
192.168.0.13:4
#192.168.0.14
```

Dla porównania – superkomputer Cray-1 z 1975 roku miał 160 MFLOPSów, a więc każde Raspberry PI 3 w klastrze to tak jakby jeden dodatkowy Cray-1. To niewiele, jak na obecne czasy, ale jest jeszcze jeden aspekt: układy Broadcoma użyte w RPI także zawierają jednostki GPU.

Jednostka do obliczeń graficznych świetnie sprawdza się w operacjach zmiennoprzecinkowych. Ta zastosowana w Raspberry PI 1 i 2 ma moc obliczeniową na poziomie 24 GFLOPS, a ta z Raspberry PI 3 – 28,8 GFLOPS. Oznacza to, że użycie GPU w Raspberry PI pozwala zwiększyć wydajność modułu tworzonego przez nas superkomputera 600-krotnie, a każdy Raspberry PI 3 dodany do klastra będzie odpowiadać 180 superkomputerom Cray-1. Ba! W pojedynczym Raspberry PI pierwszej generacji drzemie ponad dwukrotnie większa moc obliczeniowa niż w superkomputerze IBM Deep Blue, znanym z tego ze pokonał on Garriego Kasparowa w szachy. 100 starych już Raspberry PI pierwszej generacji ma teoretycznie sumaryczną moc obliczeniową na poziomie 500. superkomputera w rankingu najlepszych komputerów na świecie w 2002 roku. A zaprezentowany w dalszej części sposób można równie dobrze wykorzystać w standardowych komputerach i z pomocą nowoczesnych kart graficznych uzyskać znacznie lepsze rezultaty.



Rysunek 3. Hierarchia w systemie superkomputera

Listing 2. Program obliczający liczbę PI na superkomputerze

```
#include „mpi.h”
#include <stdio.h>
#include <math.h>

double f(double);
double f(double a)
{
    return (4.0 / (1.0 + a*a));
}

int main(int argc,char *argv[])
{
    int    n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int    namelen;
    char   processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);
    fprintf(stdout,”Process %d of %d is on %s\n”, myid, numprocs, processor_
name);
    fflush(stdout);
    n = 10000;            /* domyślna liczba prostokątów */
    if (myid == 0) startwtime = MPI_Wtime();
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    h   = 1.0 / (double) n;
    sum = 0.0;
    for (i = myid + 1; i <= n; i += numprocs)
    {
        x = h * ((double)i - 0.5);
        sum += f(x);
    }
    mypi = h * sum;
    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if (myid == 0) {
        endwtime = MPI_Wtime();
        printf(„pi is approximately %.16f, Error is %.16f\n”, pi, fabs(pi -
PI25DT));
        printf(„wall clock time = %f\n”, endwtime-startwtime);
            fflush(stdout);
    }
    MPI_Finalize();
    return 0;
}
```

Architektura superkomputera

Superkomputer różni się od zwykłego komputera tym, że wykonuje operacje za pomocą wielu różnych rdzeni. Co prawda nowoczesne procesory są już same w sobie wielordzeniowe, ale nowoczesne superkomputery są po prostu wieloprocesorowe. W gruncie rzeczy nowoczesne mikroprocesory są tak jakby scalonymi superkomputerami i optymalne wykorzystanie ich mocy obliczeniowej wymaga odpowiedniego uwzględnienia tego faktu podczas programowania. Natomiast nowoczesne superkomputery nie są całe scalone na pojedynczych krzemowych płytkach podłożowych, gdyż wykonanie takich podzespołów byłoby bardzo trudne, kosztowne (problem drastycznego spadku uzysku produkcyjnego wraz ze zwiększaniem powierzchni pojedynczego układu scalonego) oraz trudno byłoby odprowadzić z nich ciepło. Dlatego superkomputery od lat składają się z wielu procesorów, umieszczonych na kartach procesorowych lub płytach głównych i połączonych ze sobą za pomocą szybkich interfejsów.

W superkomputerze można zazwyczaj wyróżnić jednostkę nadrzędną, która rozdziela zadania na poszczególne procesory lub bloki procesorów. Te natomiast przypisują wykonywane zadania do konkretnych rdzeni. W nowoczesnych superkomputerach bywa tak, że w blokach procesorów wydzielone są jednostki, które zajmują się tylko przekazywaniem i rozdzielaniem zadań, albo np. stanowią rezerwę, na wypadek uszkodzenia któregoś rdzenia procesora – by w łatwy sposób można było utrzymać przewidziane tempo prac. Natomiast połączenia pomiędzy jednostkami w superkomputerach są wykonywane najczęściej za pomocą łączy optycznych, by nie spowalniać komunikacji. Ogólna struktura typowego superkomputera została zaprezentowana na **rysunku 3**. Nasz superkomputer również będzie zbudowany w ten sposób, z tym że do połączenia ze sobą jednostek wykorzystamy sieć ethernetową, opartą o przewody miedziane.

Jednostki wykonawcze

Praca superkomputera opiera się o wykonywanie instrukcji przez jednostki wykonawcze, którym to instrukcje zlecane są przez jednostkę nadrzędną. Ponieważ wszystkie Raspberry PI są do siebie bardzo podobne, raz przygotowane oprogramowanie będzie mogło posłużyć za wzór dla wszystkich jednostek wykonawczych tworzonego superkomputera.

Wymianę danych pomiędzy jednostkami komputera oprzemy o standard MPI (Message Passing Interface), a dokładniej o jego rozbudowaną odmianę: MPI-2. W środowisku Linux protokół MPI zaimplementowano m.in. w postaci darmowego pakietu mpich. Jest on dostępny w repozytoriach Raspbiana w całkiem nowej wersji: **mpich-3.1**.

Najnowsza dostępna wersja: **mpich-3.2** ma dodatkowo wsparcie dla Fortrana 2008, ale nie będzie ono nam potrzebne. Słowo wyjaśnienia należy się dostępności pakietów **mpich-3.1** i **mpich2-3.1** w repozytorium. Dawniej **mpich2** było implementacją MPI-2, a **mpich** jedynie MPI, jednakże od wersji 3.0, **mpich** i **mpich2** są praktycznie identyczne.

Przygotowanie wzorcowej jednostki wykonawczej nie jest trudne. Należy pobrać obraz Raspbiana – warto sięgnąć po wersję lite i nagrać ją na kartę SD. W naszym przypadku jest to plik **2016-03-18-raspbian-jessie-lite.img**. Następnie należy zalogować się do systemu i skonfigurować go zgodnie z potrzebami. Można np. ustawić dane odnośnie podstawowego języka systemowego, lokalizacji, strefy czasowej, klawiatury itd. Wypada też rozszerzyć system plików, gdyż kompletne środowisko **mpich** wraz ze wszystkimi bibliotekami zajmuje dosyć dużo miejsca. Problem jednak w tym, że szybkie rozszerzenie systemu plików z użyciem opcji w menu **raspi-config** powoduje automatyczne zajęcie całej dostępnej przestrzeni na karcie. W konsekwencji, wszystkie kolejne jednostki wykonawcze, bazujące na tworzonym w tym momencie wzorcu również będą musiały mieć karty o przynajmniej tej samej pojemności. Dodatkowo kopiowanie dużych obrazów będzie bardziej czasochłonne.

Następnym krokiem powinna być instalacja pakietu **mpich**. O ile można go samodzielnie skompilować i uruchomić, dostępność gotowych plików dla Raspbiana znacząco ułatwia sprawę oraz skraca czas instalacji. Wystarczy tylko zaktualizować informacje o pakietach w repozytorium za pomocą polecenia:

```
sudo apt-get update
```

a następnie zainstalować dwa pakiety i powiązane z nimi biblioteki poleceniem:

```
sudo apt-get install
mpich2 mpich2python
```

Całość zajmuje trochę czasu, ze względu na dużą liczbę bibliotek. Można też od razu zainstalować dodatkowe narzędzia potrzebne do wykonywania przewidywanych programów.

Przygotowaną jednostkę wykonawczą należy bezpiecznie wyłączyć (**sudo poweroff**), wyjąć z niej kartę SD i utworzyć na jej podstawie obraz w postaci pliku, który łatwo będzie można kopiować na pozostałe jednostki. To właśnie ten obraz będzie stanowił wzorzec. Przygotowanie kolejnych jednostek wykonawczych będzie polegało przede wszystkim na zgraniu obrazu na kolejne karty SD.

Jednostka sterująca

Zgodnie z **rysunkiem 1**, nad jednostkami wykonawczymi panuje jednostka sterująca. Może nią być dowolny komputer z zainstalowanym pakietem **mpich** w wersji kompatybilnej z tą zainstalowaną na jednostkach wykonawczych. W naszym przypadku, dla ułatwienia, jednostka sterującą również będzie Raspberry PI. Co więcej – ze względu na ograniczone zasoby, w praktyce rolę sterowania będzie pełnił jeden z komputerów będących także jednostką wykonawczą. Nie jest to optymalna konfiguracja dla dużych systemów, zbudowanych z identycznych komputerów, ale fakt że wszystkie komputery połączone są w klaster za pomocą sieci Ethernet sprawia, że taka konfiguracja jest w pełni możliwa.

Uzupełnienie funkcji wybranej jednostki wykonawczej o funkcję jednostki sterującej jest bardzo proste dzięki temu, że jest na niej zainstalowany pakiet **mpich**. W praktyce konieczne jest przede wszystkim wygenerowanie klucza szyfrującego, który umożliwi bezproblemową autoryzację jednostki sterującej na jednostkach wykonawczych, bez potrzeby ręcznego wpisywania haseł. Taki klucz można nawet wygenerować podczas tworzenia wzorca jednostek wykonawczych, a następnie zapisać go w odpowiednim miejscu, by nie trzeba było go ręcznie kopiować na wszystkie jednostki.

Klucz należy wygenerować poleceniem:

```
ssh-keygen -t rsa
```

W trakcie generowania komputer zapyta o dodatkowe informacje, takie jak słowo kluczowe, które będzie ograniczać dostęp do klucza. Można pozostawić je puste, co nie jest zalecane ze względów bezpieczeństwa, ale ułatwia następnie korzystanie z klastra.

Jeśli klucz generowany jest na etapie tworzenia wzorca, można go wpisać do autoryzowanych kluczy poleceniem:

```
cat ~/.ssh/id_rsa.pub
>>.ssh/authorized_keys
```

Przy czym może wystąpić konieczność wcześniejszego stworzenia katalogu **/home/pi/.ssh**.

Jeśli klucz nie został zapisany we wzorcu, trzeba go następnie ręcznie wgrać do odpowiednich katalogów wszystkich jednostek wykonawczych. Można to zrobić za pomocą następującego zbitku poleceń:

```
cat ~/.ssh/id_rsa.pub |
ssh pi@192.168.0.11 „cat
>> .ssh/authorized_keys”
```

kolejno dla wszystkich adresów IP jednostek wykonawczych. Powoduje on załadowanie klucza publicznego jednostki sterującej, dzięki któremu będzie ona rozpoznawana jako autoryzowana do komunikacji przez SSH z danymi jednostkami wykonawczymi.

Warto też zmienić nazwy hostów jednostek, dla ułatwienia ich identyfikacji. Można to zrobić kolejno wywołując polecenia:

```
ssh pi@192.168.0.11 ‘sudo
echo „rpimodule001” | sudo
tee /etc/hostname’
```

dla wszystkich adresów IP jednostek wykonawczych.

Pozostaje jeszcze problem adresowania, a więc odnajdywania poszczególnych jednostek wykonawczych przez jednostkę sterującą. Przy domyślnych ustawieniach wielu routerów, numery IP sieci LAN przypisywane są dynamicznie przez DHCP z rezerwacjami czasowymi na okres kilku godzin lub dni. A to oznacza, że z czasem mogą się one zmieniać i któreś z jednostek wykonawczych mogą wypaść poza pulę powstałą przy pierwszym uruchomieniu superkomputera. Numery IP można oczywiście skonfigurować ręcznie w systemie operacyjnym, ale – jeśli tylko mają one pracować z jednym, konkretnym routerem, wygodniej będzie odnaleźć je w panelu konfiguracyjnym routera, znaleźć odpowiadające im numery MAC interfejsów sieciowych i dokonać rezerwacji stałych w DHCP routera.

Następnie należy przygotować plik, w którym wymienione będą wszystkie jednostki wykonawcze. Plik ten ma prostą strukturę: w każdej linijce wpisywany jest adres IP kolejnej jednostki wykonawczej klastra oraz opcjonalnie, po dwukropku, liczba rdzeni danej, którymi dysponuje dana jednostka. Zamiast adresów IP można podać też nazwy hostów poszczególnych jednostek. Przykład takiego pliku przedstawiono na **listingu 1**.

W końcu wystarczy upewnić się, że wszystkie jednostki są podłączone do sieci i włączone. Superkomputer jest już gotowy.

Wykonywanie programów

Uruchamianie programu na superkomputerze polega na odpowiednim uruchomieniu go na jednostce sterującej, tak by ona przekazała kod do przetwarzania jednostkom wykonawczym. Służy temu polecenie **mpiexec**. Jednakże nie każdy program zyska na pracy na superkomputerze. Ważne jest, by został on odpowiednio napisany i skompilowany. Przykłady takich programów można znaleźć w ramach źródeł pakietu **mpich**. Można je pobrać z Internetu, najlepiej w wersji identycznej, jak zainstalowana na Raspberry PI, a więc 3.1. Plik w formacie .tar.gz ze źródłami znajduje się pod adresem: *http://goo.gl/ySS4pm*.

Po pobraniu i rozpakowaniu go (komendy **wget** oraz **tar -xzf**) w katalogu **mpich-3.1/examples/** będzie można znaleźć kody źródłowe różnych przykładowych programów. Jednym z nich jest program **cpi**, służący do obliczania liczby pi (jego kod źródłowy został pokazany na **listingu 2**). Na potrzeby uwypuklenia zalet superkomputera,

edytowaliśmy go, by zmienić dokładność jego pracy, podmieniając wartość n=10000 na n=1000000.

Kod należy następnie skompilować, ale zamiast użyć polecenia **gcc**, używamy **mpicc**, również zainstalowanego wraz z pakietem **mpich**. Kompilacja wykonywana jest więc wywołując:

```
cd ./mpich-3.1/examples/
mpicc cpi.c -o cpi
```

W wyniku powyższych operacji, w katalogu z przykładami otrzymujemy plik wykonywalny **cpi**.

Można go następnie zwyczajnie wywołać:

```
./mpich-3.1/examples/cpi
```

co w naszym przypadku powoduje wyświetlenie informacji, że czas wykonywania programu wyniósł ok 0,110 s (**rysunek 4**). Jednakże aby wykonać go nie na jednostce sterującej, ale na całym superkomputerze, należy skorzystać z polecenia **mpiexec**, wskazując plik z listą jednostek wykonawczych, liczbę jednostek które mają brać udział w wykonywaniu pracy oraz właściwy plik wykonywalny do przetwarzania. W przypadku dwóch jednostek polecenie będzie wyglądało następująco:

```
mpiexec -f machinefile -n 2 ./
mpich-3.1/examples/cpi
```

parametr **–f** pozwala wskazać plik z listą jednostek, a parametr **–n** umożliwia podanie liczby procesów, na które podzielony ma być plik wykonywany. Efekt uruchomienia programu na dwóch jednostkach wykonawczych (z czego jedna pełni jednocześnie rolę jednostki sterującej) widać na **rysunku 5**. Sumaryczny czas pracy zmalał do 0,064 s, czyli prawie dwukrotnie. Można również wymusić podział pracy na większą liczbę procesów, pomimo posiadania jedynie dwóch jednostek wykonawczych. System superkomputera wtedy samodzielnie zdecyduje (w sposób bardzo deterministyczny), jak rozdzielić pracę, opierając się o informacje o dostępnych rdzeniach na poszczególnych jednostkach. Jeśli każda jednostka ma tylko jeden rdzeń, prace zostaną podzielone po równo (**rysunek 6**). Jeśli jednak zadeklarowaliśmy w pliku **machinefile**, że jedna jednostka ma 3 rdzenie, a druga tylko jeden, w przypadku podziału na cztery procesy, jednostka 3-rdzeniowa dostanie dokładnie trzy procesy do wykonania, a 1-rdzeniowa tylko jeden (**rysunek 7**).

Oglądając **obrazki 4 i 5** warto zauważyć, że czas pracy dla dwóch jednordzeniowych jednostek wykonawczych, które otrzymają więcej niż jeden proces do przetwarzania, będzie dłuższy niż w przypadku optymalnym, pokazanym na **rysunku 3**. Przełączanie wątków procesora oraz przesyłanie danych przy równym podziale zwiększyło czas wykonania programu do ok. 0,078 s, a przy nierównomiernym obciążeniu, aż do 0,092 s. Jednocześnie zmieniła się precyzja obliczenia liczby pi, co wynika

```
pi@raspberrypi101:~/mpitest $ ./mpich-3.1/examples/cpi
Process 0 of 1 is on raspberrypi101
pi is approximately 3.1415926535897643, Error is 0.0000000000000289
wall clock time = 0.109544
```

Rysunek 4. Scr2 – program wykonany samodzielnie na jednym RPI

```
pi@raspberrypi101:~/mpitest $ mpiexec -f machinefile -n 2 ./mpich-3.1/examples/cpi
Process 0 of 2 is on raspberrypi101
Process 1 of 2 is on raspberrypi102
pi is approximately 3.1415926535899388, Error is 0.0000000000001457
wall clock time = 0.064366
```

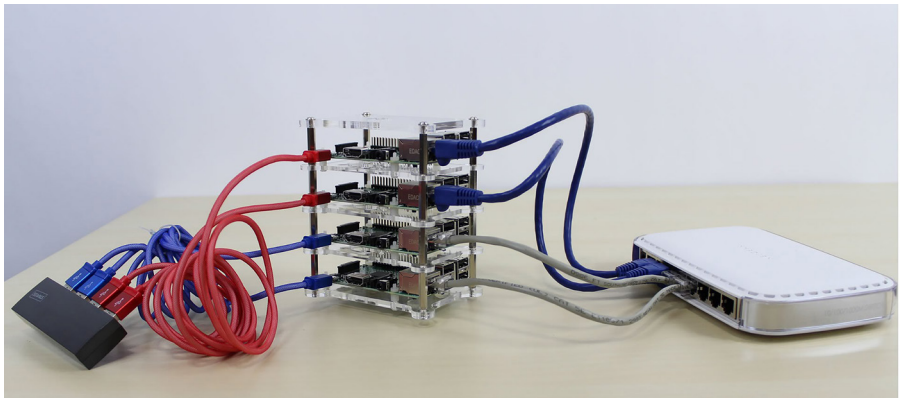
Rysunek 5. Scr1 – program wykonany na dwóch RPI – po jednym wątku na komputer

```
pi@raspberrypi101:~/mpitest $ mpiexec -f machinefile -n 4 ./mpich-3.1/examples/cpi
Process 3 of 4 is on raspberrypi102
Process 1 of 4 is on raspberrypi102
Process 0 of 4 is on raspberrypi101
Process 2 of 4 is on raspberrypi101
pi is approximately 3.1415926535899028, Error is 0.0000000000001097
wall clock time = 0.077925
```

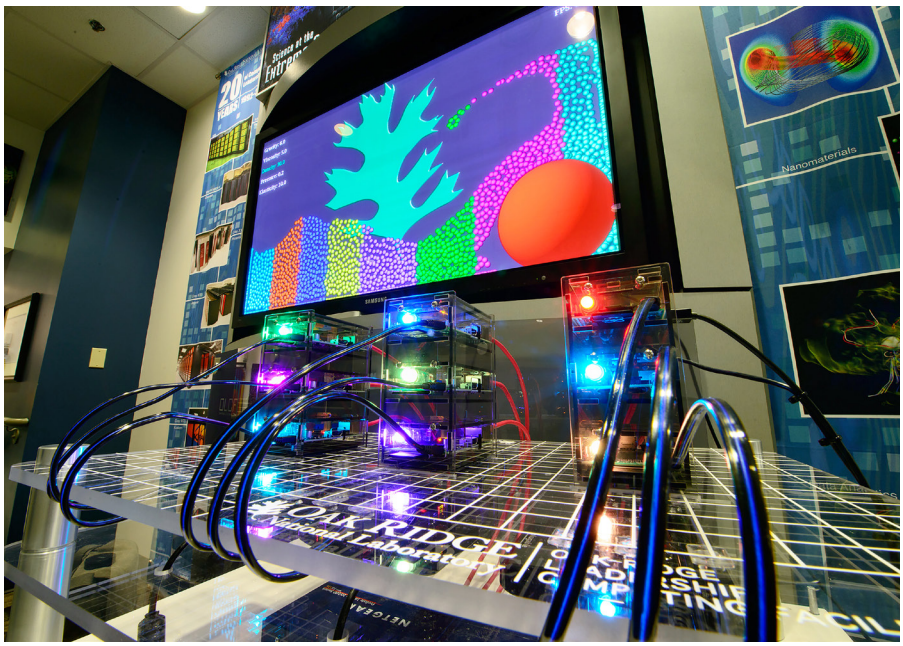
Rysunek 6. Scr3 – program wykonany na dwóch RPI – po dwa wątki na komputer

```
pi@raspberrypi101:~/mpitest $ mpiexec -f machinefile -n 4 ./mpich-3.1/examples/cpi
Process 3 of 4 is on raspberrypi102
Process 2 of 4 is on raspberrypi101
Process 1 of 4 is on raspberrypi101
Process 0 of 4 is on raspberrypi101
pi is approximately 3.1415926535899033, Error is 0.0000000000001101
wall clock time = 0.092222
```

Rysunek 7. Scr4 – program wykonany na dwóch RPI – 3 wątki na jednym i 1 na drugim



Fotografia 8. 4-elementowy superkomputer z Raspberry PI, zbudowany przez Rasima Muratovica z Nowego Jorku



Fotografia 9. Tiny Titan to 9-elementowy superkomputer z Raspberry PI, przygotowany na potrzeby nauczania programowania superkomputerów

Fotografia 10. Superkomputer złożony ze 120 Raspberry PI, wyposażone w łącznie 120 modułów z wyświetlaczami

ze specyfiki sposobu pisania programów na superkomputery.

Tabela 2. Lista podstawowych funkcji środowiska mpich dla języków C i Fortran

Funkcja	Wywołanie w języku C	Wywołanie w języku Fortran
Inicjalizacja środowiska superkomputera	int MPI_Init(int *argc, char **argv)	integer ierror call MPI_Init(ierror)
Zapytanie o liczbę jednostek wykonawczych w klastrze	int MPI_Comm_size(MPI_Comm comm, int *size)	integer comm,size,ierror call MPI_Comm_Size(comm,size,ierror)
Zapytanie o numer jednostki wykonawczej, przypisany procesorowi, który wywołuje tę funkcję	int MPI_Comm_rank(MPI_Comm comm, int *rank)	integer comm,rank,ierror call MPI_Comm_Rank(comm,rank,ierror)
Zakończenie pracy klastra nad danym zadaniem	int MPI_Finalize()	CALL MPI_Finalize(ierror)
Wysłanie wiadomości, za pomocą której przekazywane są dane pomiędzy jednostkami	int MPI_Send (void *buf,int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)	<type> buf(*) integer count, datatype,dest,tag integer comm, ierror call MPI_Send(buf,count, datatype, dest, tag, comm, ierror)
Odebranie wiadomości z danymi z innej jednostki wykonawczej	int MPI_Recv (void *buf,int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)	<type> buf(*) integer count, datatype, source,tag integer comm, status, ierror call MPI_Recv(buf,count, datatype, source, tag, comm, status, ierror)

Programowanie superkomputera

Aby napisany program był w stanie skorzystać z możliwości superkomputera, musi przewidywać podział pracy na wiele jednostek wykonawczych. Zamiast wykonywać cały algorytm krok po kroku, trzeba go podzielić na bloki, które zostaną przekazane do wykonania innym jednostkom.

Nie każdy program da się w ten sposób łatwo zmodyfikować, ale szczęśliwie, większość zadań przetwarzania danych można wykonywać równolegle – tak samo jak np. renderuje się grafikę na nowoczesnych GPU, które same w sobie mają bardzo wiele rdzeni. Wystarczy tylko podzielić dane na bloki i przypisać je do poszczególnych jednostek wykonawczych lub rdzeni. Następnie wynik obliczeń musi być przesłany w jedno miejsce i scalony.

W przypadku środowiska **mpich**, w języku C do podziału i przekazywania danych wykorzystywana jest biblioteka **mpi.h**. Pozwala ona m.in. wewnątrz programu, dowiedzieć się, ile jest jednostek wykonawczych w klastrze, jaki jest numer danej jednostki, która aktualnie przetwarza przygotowany kod oraz wysyłać dane do innych jednostek. Lista podstawowych funkcji dla języków C i Fortran została zebrana w **tabeli 2**.

Na początku programu konieczna jest inicjalizacja środowiska. Przydatne może być sprawdzenie liczby jednostek w klastrze – pozwala to określić w jaki sposób jednostki mają dzielić między siebie zadania. Funkcja **MPI_Comm_rank()** umożliwia natomiast zdobycie informacji o aktualnym numerze jednostki, która także może posłużyć do podziału zadania. Przykładowo, jeśli dostępne jest N jednostek, a do przetworzenia jest blok danych o długości X segmentów, program może decydować, że na danej jednostce przetworzy jedynie X/N segmentów, a dokładniej tylko te, dla których warunek:

```
if ((x mod N) == K)
```

jest spełniony (gdzie x to numer aktualnego segmentu, a K to numer danej jednostki wykonawczej.

Na koniec działania jednostek wykonawczych należy wywołać funkcję **MPI_Finalize()**, które informuję jednostkę sterującą, że dana jednostka wykonawcza ukończyła przyznane jej zadanie. Program kończy pracę, gdy wszystkie jednostki wykonawcze zgłoszą koniec prac.

Natomiast przesyłanie danych wymaga wskazania przede wszystkim bufora, liczby i rodzaju danych, jednostki źródłowej lub docelowej oraz kilku dodatkowych informacji. W wywołaniach funkcji pojawia się też odniesienie do zmiennej **MPI_COMM_WORLD**, która jest definiowana wewnątrz funkcji **MPI_init()**, a jej typ został określony w bibliotece. Analogiczny zestaw funkcji można znaleźć dla języka Python.

Marcin Karbowniczek, EP

Fotografia 11. Superkomputer Titan, zbudowany przez firmę Cray i zainstalowany w Oak Ridge National Laboratory, złożony z procesorów AMD Opteron, połączonych z jednostkami GPU Nvidia Tesla