INNOWACYJNA GOSPODARKA
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ
ROZWOJU REGIONALNEGO

Projekt nr POIG.02.03.00.24-093/13
DOTACJE NA INNOWACJE
Projekt współfinansowany ze środków Europejskiego Funduszu Rozwoju Regionalnego w ramach Programu Operacyjnego Innowacyjna Gospodarka

INSTRUKCJA UŻYTKOWNIKA PROJEKTU MICLAB

Wersja 0.2.5

Spis treści

1 Wstęp **2**

1.1 Projekt $MICLAB$ 2

2 Zasoby sprzętowe i programowe **3**

2.1 Opis sprzętu 3

2.2 Oprogramowanie 3

3 Proces zakładania konta **5**

4 Logowanie **6**

4.1 Logowanie w systemie Linux 6

4.1.1 Przekierowanie wyświetlania 7

4.2 Logowanie w systemie MS Windows 8

4.2.1 Przekierowanie wyświetlania 10

4.3 Dostępne węzły obliczeniowe 11

5 Pierwsze kroki **12**

6 Konfiguracja środowiska **15**

7 Pierwszy program - tryb natywny **16**

8 Uruchamianie aplikacji w modelu offload **19**

9 Wykorzystanie biblioteki MKL na koprocesorach Intel® Xeon Phi™ **22**

9.1 Tryb natywny MKL 22

9.2 Automatyczny tryb offload MKL 24

1 Wstęp

Nie ulega wątpliwości, że nowoczesne systemy masywnie wielordzeniowe są coraz to bardziej atrakcyjne dla prac naukowo-badawczych prowadzonych w wiodących ośrodkach. Dzięki nim staje się możliwe przeprowadzanie badań o większym wymiarze, pozwalających na znalezienie odpowiedzi na pytania nurtujące naukowców, a będące wcześniej poza zasięgiem ich możliwości.

Jednakże problemy wynikające z użytkowania nietradycyjnych systemów obliczeniowych, jakimi są koprocesory Intel Xeon Phi, nie zachęcają społeczności badawczo-naukowej do ich stosowania. Brak łatwego i szybkiego dostępu do nowoczesnych platform tego typu, brak eksperckiej wiedzy, brak gwarancji korzyści z ich użycia, jak również brak wsparcia dla użytkowników jedynie potęgują ten efekt.

Dlatego też niesłychanie ważne staje się utworzenie laboratorium pilotażowego systemów masywnie wielordzeniowych, które ma na celu ułatwienie oraz zachęcenie społeczności naukowo-badawczej do korzystania z takich rozwiązań. Testowanie infrastruktury obliczeniowej w warunkach laboratoryjnych zdecydowanie ułatwi zdobycie gruntownej wiedzy w zakresie ich prawidłowego wykorzystywania.

1.1 Projekt $MICLAB$

Głównym celem projektu $MICLAB$ jest udostępnienie elektronicznego laboratorium, czyli wirtualnego miejsca, w którym ogólnokrajowa społeczność naukowo-badawcza będzie mogła zbadać możliwości wykorzystania i określić kierunki zastosowania najnowszych, masywnie wielordzeniowych architektur obliczeniowych w wiodących dziedzinach nauki.

Bezpośrednim celem projektu jest budowa pilotażowej instalacji obliczeniowej opartej o masywnie wielordzeniowe koprocesory obliczeniowe najnowszej generacji typu Intel® Xeon Phi™, a także wsparcie dla możliwie maksymalnego wykorzystania tej nowej, nietradycyjnej architektury obliczeniowej przez jak najszersze grono użytkowników w naszym kraju.

2 Zasoby sprzętowe i programowe

2.1 Opis sprzętu

W chwili obecnej w projekcie $MICLAB$ udostępniane są cztery węzły obliczeniowe, do których dostęp jest realizowany przez węzeł dostępowy. Maszyny te połączone są ze sobą siecią InfiniBand FDR 56Gb/s przy wykorzystaniu przełącznika Mellanox MSX6015F-1SFS. Poniżej znajduje się specyfikacja poszczególnych wezłów projektu $MICLAB$:

- węzeł obliczeniowy:

Platforma:	Serwer Actina Solar 220 X5 (Intel R2208GZ4GC Grizzly Pass)
CPU:	2 x Intel Xeon E5-2695 v2 (2x12 rdzeni, 2.4 GHz)
Pamięć:	128GB ECC Registred 1866MHz (16 x 8GB)
Karty sieciowe:	2x InfiniBand: Mellanox MCB191A-FCAT (Connect-IB)
Koprocesor:	2x Intel Xeon Phi Coprocessor 7120P

- węzeł dostępowy:

Platforma:	Serwer Actina Solar G 210 S5 (Supermicro 1027GR-TSF)
CPU:	2 x Intel Xeon E5-2665 (2x8 rdzeni, 2.4 GHz)
Pamięć:	64GB ECC Registred 1600MHz (8 x 8GB)
Karty sieciowe:	1x InfiniBand: Mellanox MCB191A-FCAT (Connect-IB)

2.2 Oprogramowanie

Wezły projektu $MICLAB$ pracują pod kontrolą systemu operacyjnego Linux. W tej chwili dostępne jest na nich następujące oprogramowanie:

- Intel Parallel Studio XE Cluster Edition for Linux, który składa się z:
  - Intel Advisor XE for Linux
  - Intel HD Graphics Drivers for Linux
  - Intel Inspector XE for Linux

- Intel Integrated Performance Primitives for Linux
- Intel Math Kernel Library for Linux
- Intel MPI Library for Linux
- Intel MPSS for Linux
- Intel Parallel Studio XE Composer Edition for C++ Linux
- Intel Parallel Studio XE Composer Edition for Fortran Linux
- Intel Threading Building Blocks for Linux
- Intel Trace Analyzer and Collector for Linux
- Intel VTune™ Amplifier XE - OS X host only
- Intel VTune™ Amplifier XE for Linux

3 Proces zakładania konta

Do uzupełnienia przez Marcina

4 Logowanie

Po założeniu konta przez administratora możemy przejść do zalogowania się na maszyny z koprocesorami Intel® Xeon Phi™ działające w ramach projektu $MICLAB$. Do logowania używamy nazwy użytkownika oraz hasła, podanego podczas tworzenia konta w portalu internetowym $MICLAB$. W chwili obecnej logowanie jest możliwe tylko przy użyciu protokołu ssh (wersja 2). Obecnie zasoby obliczeniowe projektu $MICLAB$ nie są zarządzane przy użyciu systemu kolejkowego (zostanie on wprowadzony w późniejszym czasie). Logując się na adres `miclab.pl` zostajemy przekierowani na pierwszy z węzłów obliczeniowych zawierający koprocesory Intel® Xeon Phi™. Wszystkie maszyny projektu $MICLAB$ są wyposażone w system operacyjny Linux, w związku z tym do prowadzenia obliczeń niezbędna jest znajomość podstawowych poleceń tekstowych tego systemu.

4.1 Logowanie w systemie Linux

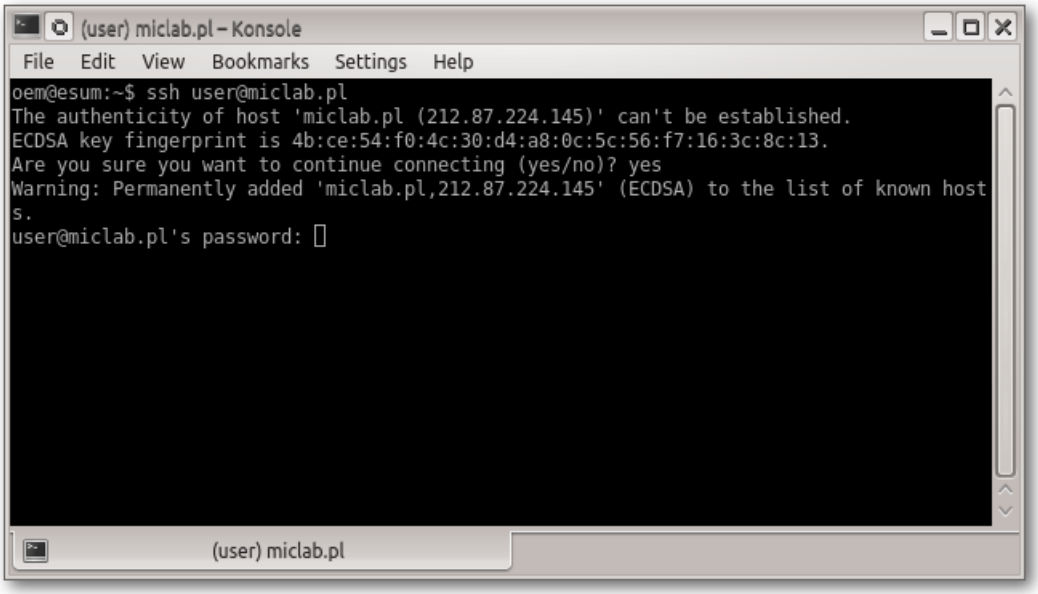
W systemie Linux pierwszą rzeczą, którą musimy zrobić jest uruchomienie terminala lub konsoli systemowej. Najczęściej ta aplikacja jest identyfikowana przez poniższą lub podobną ikonę:

Aby można było nawiązań połączenie z maszynami projektu $MICLAB$ potrzebny jest klient ssh (zaleca się korzystać z pakietu OpenSSH - jeżeli go nie ma w systemie, to należy go zainstalować korzystając z odpowiednich narzędzi systemowych zależnych od wykorzystywanej dystrybucji Linuxa). Polecenie służące do zalogowania użytkownika w systemie Linux może wyglądać następująco:

```
ssh user@miclab
```

gdzie `user` jest nazwą użytkownika, jaką podaliśmy przy tworzeniu konta w portalu internetowym $MICLAB$. Najczęściej przy pierwszym logowaniu zostajemy poproszeni o potwierdzenie

klucza dostępu - należy wpisać `Yes`. Następnie podajemy hasło, które również jest takie samo jak dla portalu:



Jeżeli logowanie zakończyło się sukcesem, to w oknie terminala powinna zostać wyświetlona linia podobna do poniższej:

```
[user@MICLAB-01 ~]$
```

gdzie zamiast `user` będzie umieszczona właściwa nazwa użytkownika, jaką użyliśmy do logowania.

4.1.1 Przekierowanie wyświetlania

Jeżeli podczas pracy zdalnej chcemy korzystać z aplikacji działających w trybie graficznym, to musimy włączyć przekierowanie wyświetlania (przez protokół ssh będą wówczas również przesyłane komunikaty systemu X Window). W obecnych dystrybucjach systemu Linux jedyne, co może być do tego potrzebne, to dodatkowe użycie opcji `-X` lub `-Y` podczas nawiązywania połączenia (nie jest potrzebne ręczne ustawianie zmiennej `DISPLAY` w sytemie docelowym). Przykład użycia:

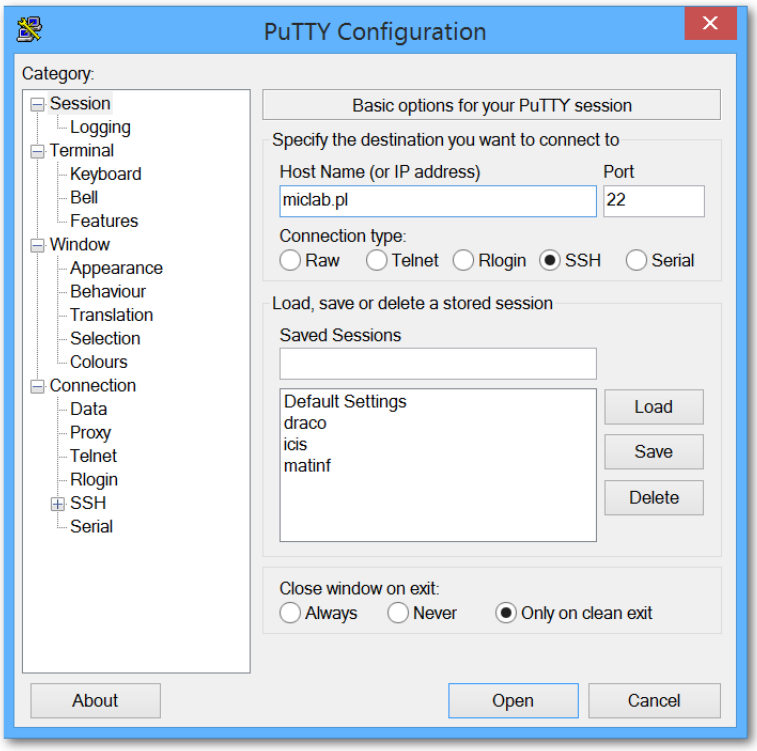
```
ssh -X  user@miclab
```

4.2 Logowanie w systemie MS Windows

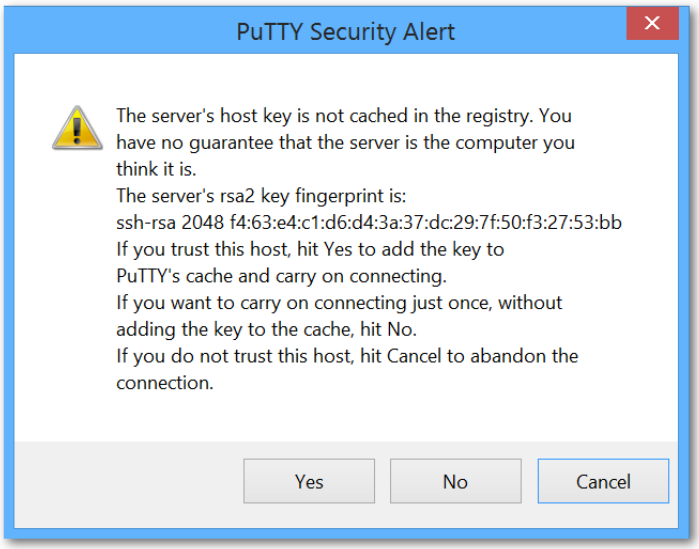
W systemie Windows nie ma standardowo zainstalowanego klienta dla protokołu ssh. Możemy do tego celu wykorzystać jedno z darmowych rozwiązań, jakim jest program `PuTTY`. Plik wykonywalny można pobrać z adresu:

`http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe`

Po uruchomieniu ściągniętego programu ukazuje się następujące okienko:



W polu **Host Name (or IP address)** wpisujemy `miclab.pl` i klikamy w przycisk **Open**. Przy pierwszym logowaniu wyświelone zostanie okienko z komunikatem dotyczącym klucza dostępu. Należy zatwierdzić klucz poprzez kliknięcie przycisku **Yes**:



Następnie zostanie wyświetlone okno terminalu, w którym zostaniemy poproszeni o wpisanie nazwy użytkownika i hasła (są one takie same, jak w przypadku portalu internetowego projektu $MICLAB$):

Jeżeli logowanie zakończyło się sukcesem, to w oknie terminala powinna zostać wyświetlona linia podobna do poniższej:

```
[user@MICLAB-01 ~]$
```

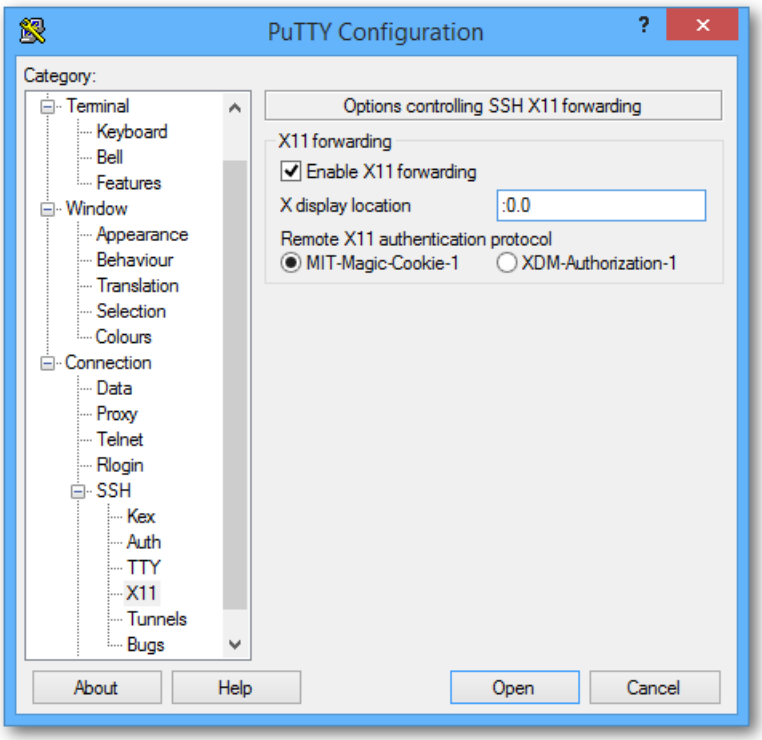
gdzie zamiast `user` będzie umieszczona właściwa nazwa użytkownika, jaką użyliśmy do logowania.

4.2.1 Przekierowanie wyświetlania

W przypadku systemu MS Windows również możemy korzystać z aplikacji zdalnych działających w trybie graficznym. Ponieważ system ten standardowo nie posiada serwera wyświetlania systemu X Window, to najpierw musimy go zainstalować. Do tego celu możemy wykorzystać jedno z darmowych rozwiązań jakim jest Xming. Instalator możemy pobrać z adresu:

`http://sourceforge.net/projects/xming/`

Instalujemy program oraz uruchamiamy go, a następnie przystępujemy do uruchomienia i konfiguracji klienta ssh (`PuTTy`). Po wprowadzeniu nazwy hosta (`miclab.pl`), przechodzimy do kategorii `Connection > SSH > X11`:



Zaznaczamy opcję **Enable X11 forwarding**, w pole **X display location** wprowadzamy „`:0.0`”, a następnie klikamy przycisk **Open**.

Należy pamiętać, że przed każdym uruchomieniem aplikacji działających w trybie graficznym, w przypadku systemu MS Windows, należy uruchomić serwer wyświetlania systemu X Window (w naszym przypadku Xming).

4.3 Dostępne węzły obliczeniowe

Aktualnie w projekcie $MICLAB$ dostępne są cztery węzły obliczeniowe zawierające po dwa koprocesory Intel® Xeon Phi™ na węzeł. Obecnie po zalogowaniu zostajemy przekierowani na pierwszy węzeł obliczeniowy. Wszystkie węzły obliczeniowe korzystają ze wspólnego systemu plików NFS, w ramach którego katalogi domowe użytkowników są współdzielone. Może zdarzyć się sytuacja, że inna osoba korzysta również z tego węzła lub chcielibyśmy wykorzystać większą liczbę węzłów obliczeniowych. Do identyfikacji poszczególnych węzłów możemy skorzystać z ich nazw lub nazw skróconych, które zostały umieszczone w poniższej tabeli:

numer węzła	nazwa pełna	nazwa skrócona
1	node1	n1
2	node2	n2
3	node3	n3
4	node4	n4

Przykładowo, aby zalogować się na inny węzeł obliczeniowy używamy polecenia `ssh` oraz podajemy jego nazwę. Na przykład do zalogowania się na węzeł numer 2 możemy użyć polecenia:

```
[user@MICLAB-01 ~]$  ssh  node2
```

lub:

```
[user@MICLAB-01 ~]$  ssh  n2
```

5 Pierwsze kroki

Po zalogowaniu na węzeł obliczeniowy możemy sprawdzić, ile koprocesorów Intel® Xeon Phi™ jest na nim zainstalowanych i dostępnych. W tym celu można posłużyć się poleceniem `micctrl` z parametrem `status`:

```
[user@MICLAB-01 ~]$ micctrl --status
mic0: online (mode: linux image: /usr/share/mpss/boot/bzImage-knightscorner)
mic1: online (mode: linux image: /usr/share/mpss/boot/bzImage-knightscorner)
```

W powyższym przykładowym wyniku otrzymujemy informację o tym, że są dostępne dwa koprocesory - `mic0` i `mic1`.

Parametry zainstalowanych koprocesorów można pobrać przy użyciu polecenia `micctrl` z opcją `-a`:

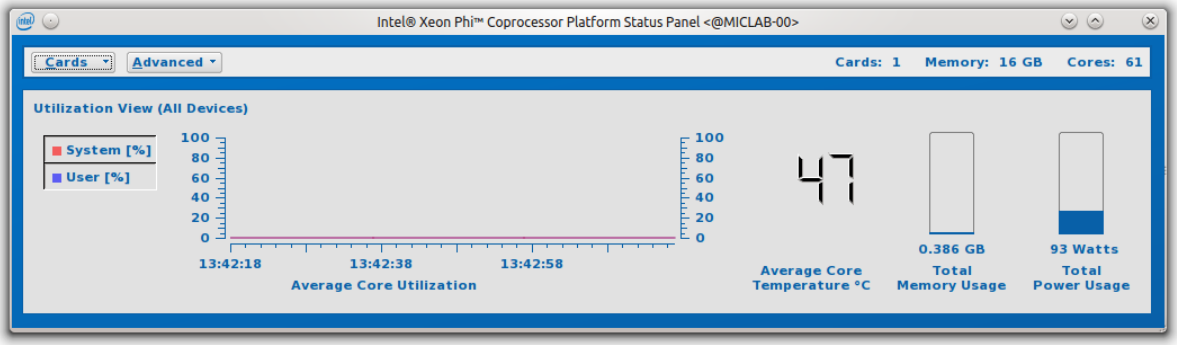
```
[user@MICLAB-01 ~]$ micsmc -a
mic0 (info):
   Device Series: ........... Intel(R) Xeon Phi(TM) coprocessor x100 family
   Device ID: ............... 0x225c
   Number of Cores: ......... 61
   OS Version: .............. 2.6.38.8+mpss3.1.2
...

mic1 (info):
   Device Series: ........... Intel(R) Xeon Phi(TM) coprocessor x100 family
   Device ID: ............... 0x225c
   Number of Cores: ......... 61
...

mic0 (temp):
   Cpu Temp: ................ 34.00 C
...

mic0 (freq):
```

```
    Core Frequency: ..... 1.24 GHz
    Total Power: ..... 94.00 Watts
...

mic0 (mem):
    Free Memory: ..... 15136.93 MB
    Total Memory: ..... 15513.18 MB
    Memory Usage: ..... 376.25 MB
...

mic0 (cores):
    Device Utilization: User:   0.00%,   System:   0.08%,   Idle:  99.91%
    Per Core Utilization (61 cores in use)
        Core #1:   User:   0.00%,   System:   0.00%,   Idle: 100.00%
        Core #2:   User:   0.00%,   System:   0.27%,   Idle:  99.73%
...
```

Z wyświetlonych danych możemy dowiedzieć się między innymi o tym, jakiego rodzaju jest dany koprocesor (`Device Series`), ile posiada rdzeni (`Number of Cores`), jaką częstotliwością są one taktowane (`Core Frequency`) oraz ile posiada pamięci (`Total Memory`).

Jeżeli mamy dostęp do interfejsu graficznego, to możemy uruchomić narzędzie *Intel® Xeon Phi™ Coprocessor Platform Status Panel* (służy do tego polecenie `micsmc`):

```
[user@MICLAB-01 ~]$ micsmc
```



Narzędzie to nie tylko wyświetla informacje dotyczące liczby zainstalowanych koprocesorów, rozmiaru pamięci i liczby rdzeni, ale również prezentuje, na bieżąco uaktualniane, informacje dotyczące obciążenia, temperatury, wykorzystania pamięci oraz poboru mocy przez poszczególne koprocesory.

6 Konfiguracja środowiska

Zanim przejdziemy do kompilacji i uruchomienia pierwszego programu musimy skonfigurować środowisko, tak aby możliwe było korzystanie z niezbędnych narzędzi. Po pierwsze należy sprawdzić, czy kompilatory firmy Intel są poprawnie skonfigurowane. W tym celu wydajemy polecenie:

```
[user@MICLAB-01 ~]$ which icc
```

Wynik uruchomienia powyższego polecenia powinien wyglądać podobnie do poniższego:

```
/opt/intel/composer_xe_2015.0.090/bin/intel64/icc
```

Jeżeli natomiast wynik polecenia będzie wyglądał następująco:

```
/usr/bin/which: no icc in (/usr/local/bin: ... ),
```

to należy uruchomić poniższe polecenie:

```
[user@MICLAB-01 ~]$ source /opt/intel/bin/iccvars.sh intel64
```

Powoduje ono ustawianie zmiennych środowiskowych dla kompilatorów emphIntel C Compiler i *Intel C++ Compiler*. Dla wygody dobrze jest dodać wywołanie tego polecenia do pliku `.bashrc`, tak aby automatycznie było ono uruchamiane po zalogowaniu:

```
[user@MICLAB-01 ~]$ echo "source /opt/intel/bin/iccvars.sh intel64" >> .bashrc
```

7 Pierwszy program - tryb natywny

Najprostszym sposobem uruchamiania zadań na koprocesorze Intel® Xeon Phi™ jest wykorzystanie do tego celu modelu natywnego. Dlatego zostanie on przedstawiony jako pierwszy. W tym modelu cały program jest kompilowany dla architektury Intel® Xeon Phi™ i nie jest konieczne wprowadzanie modyfikacji do kodu programu (modyfikacje są jednakże zazwyczaj niezbędne, gdy chcemy osiągnąć większą wydajność).

Przykładowy program, który będziemy uruchamiać przedstawia poniższy listing (nazwa pliku `first.c`):

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    printf("Liczba dostepnych logicznych rdzeni: %d\n",
           sysconf(_SC_NPROCESSORS_ONLN ));
}
```

Kompilacja i uruchomienie tego programu na procesorze hosta (w przypadku maszyn dostępnych w projekcie MICLAB są to standardowe procesory Intel Xeon) może wyglądać następująco:

```
[user@MICLAB-01 ~]$ icc first.c
[user@MICLAB-01 ~]$ ./a.out
Liczba dostepnych logicznych rdzeni:  16
```

Jeżeli chcemy skompilować program dla architektury Intel® Xeon Phi™, to musimy posłużyć się mechanizmem kompilacji skrośnej (ang. *cross-compiling*). Do tego celu służy parametr `-mmic` kompilatora Intel. Program dla koprocesora możemy skompilować na procesorze hosta, jednakże próba uruchomienia na nim takiego programu zakończy się niepowodzeniem:

```
[user@MICLAB-01 ~]$ icc -mmic first.c
[user@MICLAB-01 ~]$ ./a.out
-bash: ./a.out: cannot execute binary file
```

Koprocesor Intel® Xeon Phi™ pracuje pod kontrolą systemu operacyjnego Linux i jest oddzielnym węzłem sieciowym posiadającym swój adres sieciowy. Dlatego też jest możliwe zalogowanie się na niego przy użyciu protokołu ssh. Pliki mogą być przesyłane pomiędzy koprocesorem i systemem hosta przy wykorzystaniu polecenia scp. Do koprocesora można się odwoływać poprzez jego adres sieciowy, jednakże wygodniejszą metodą jest posługiwanie się jego nazwą (aliasem):

numer węzła	numer koprocesora	nazwa pełna	nazwa skrócona
1	0	node1-mic0	n1m0
	1	node1-mic1	n1m1
2	0	node2-mic0	n2m0
	1	node2-mic1	n2m1
3	0	node3-mic0	n3m0
	1	node3-mic1	n3m1
4	0	node4-mic0	n4m0
	1	node4-mic1	n4m1

Standardowo aby uruchomić program na koprocesorze należy skopiować utworzony wcześniej plik wykonywalny do systemu plików koprocesora (zazwyczaj używa się do tego celu komendy `scp`). Jednakże na maszynach projektu MICLAB nie musimy tego robić ponieważ katalog domowy jest wspólny dla hosta i koprocesora (wspólny zasób dyskowy zamontowany na hoście i koprocesorze przy wykorzystaniu protokołu NFS). Dlatego też, aby uruchomić, w tym trybie, program na koprocesorze wystarczy się na niego zalogować, a następnie uruchomić utworzony wcześniej na hoście plik wykonywalny skompilowany dla architektury Intel MIC. Przykładowy ciąg poleceń służących do tego celu przedstawiono poniżej (zakładamy że plik wykonywalny znajduje się w katalogu głównym użytkownika, w przeciwnym wypadku należy po zalogowaniu na koprocesor przejść do odpowiedniego katalogu):

```
[user@MICLAB-01 ~]$ icc -mmic first.c
[user@MICLAB-01 ~]$ ssh node1-mic0
[user@MICLAB-01-mic0:~]$ ./a.out
Liczba dostepnych logicznych rdzeni:  244
```

Po wyświetlonej liczbie dostępnych logicznych rdzeni możemy wywnioskować, że faktycznie program został uruchomiony na koprocesorze (61 rdzenie $\times 4$ sprzętowe wątki na rdzeń $= 244$ logicznych rdzeni).

Do uruchamiania zadań na koprocesorach możemy również wykorzystać narzędzie `micnativeloadex`. Ułatwia ono uruchamianie programów na koprocesorach ponieważ elminuje konieczność logowania się na nie. Program, który wcześniej skompilowaliśmy dla architekture Intel MIC możemy uruchomić bezpośrednio z poziomu systemu hosta:

```
[user@MICLAB-01 ~]$ micnativeloadex ./a.out
Liczba dostepnych logicznych rdzeni:  244
```

8 Uruchamianie aplikacji w modelu offload

Kompilator offload firmy Intel (offload compiler) jest heterogenicznym kompilatorem zawierającym środowiska kompilacji zarówno dla CPU hosta, jak również koprocesora Intel® Xeon Phi™. W tym trybie programista określa, które fragmenty kodu mają zostać uruchomione na procesorze hosta, a które fragmenty będą realizowane przez koprocesor. Kod ten jest kompilowany przez kompilator heterogeniczny na hoście, a następnie na nim uruchamiany. Fragmenty, które mają zostać wykonane na koprocesorze są w sposób automatyczny do niego kopiowanie i tam uruchamiane. Użytkownik w tym trybie nie musi kopiować plików na koprocesor, jak również się na niego logować, jak miało to miejsce w trybie natywnym.

Przykładowy program, wykorzystujący mechanizm realizacji fragmentu kodu na koprocesorze Intel® Xeon Phi™ w trybie offload (nazwa pliku `second.c`):

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    printf("Framgment wykonywany przez CPU: \n"
           "    Liczba dostepnych logicznych rdzeni: %d\n",
           sysconf(_SC_NPROCESSORS_ONLN ));

#pragma offload target(mic)
    {
        printf("Framgment wykonywany przez koprocesor: \n"
               "    Liczba dostepnych logicznych rdzeni: %d\n",
               sysconf(_SC_NPROCESSORS_ONLN ));
    }
}
```

Dyrektywa `#pragma offload target(mic)` oznacza, że instrukcja lub blok języka C++ występujący bezpośrednio po niej ma zostać wykonany na koprocesorze Intel® Xeon Phi™ (blok offload). Przykład kompilacji i uruchomienia powyższego kodu może wyglądać następująco:

```
[user@MICLAB-01 ~]$ icc second.c
[user@MICLAB-01 ~]$ ./a.out
Framgment wykonywany przez CPU:
    Liczba dostepnych logicznych rdzeni:  16
Framgment wykonywany przez koprocesor:
    Liczba dostepnych logicznych rdzeni:  244
```

Do koprocesora jest kopiowany z hosta nie tylko kod programu, ale również dane niezbędne do jego wykonania. Zasada jest następująca: wszystkie zmienne wykorzystywane w bloku offload, a zadeklarowane poza nim są przed wykonaniem tego bloku kopiowane na koprocesor, a po jego zakończeniu są z powrotem kopiowane do hosta. Ponadto programista może przy użyciu klauzul `in`, `out`, `inout` dyrektywy `offload` sterować kierunkiem przesyłania danych pomiędzy koprocesorem i hostem. Poniżej przedstawiono przykładowy kod, w którym wykorzystano taką klauzulę (`third.c`):

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    long nCores = 0;

#pragma offload target(mic) in(nCores)
    {
        printf("Koprocesor: wartosc zmiennej nCores = %d\n", nCores);
        fflush(0);
        nCores = sysconf(_SC_NPROCESSORS_ONLN );
    }

    printf("Host: wartosc zmiennej nCores = %d\n", nCores);
}
```

W powyższym kodzie użyto klauzuli `out` dla zmiennej `nCores`. W związku z tym po wejściu do bloku offload jej wartość jest nieokreślona. W koprocesorze przypisana jej zostaje wartość

równa liczbie jego logicznych rdzeni. Po zakończeniu bloku offload wartość zmiennej `nCores` zostaje przesłana do hosta, w którym zostaje ona wyświetlona przy użyciu funkcji `printf`. Wynik działania powyższego programu będzie wyglądać następująco:

```
Koprocesor:  wartosc zmiennej nCores = 0

Host:  wartosc zmiennej nCores = 244
```

9 Wykorzystanie biblioteki MKL na koprocesorach Intel® Xeon Phi™

Math Kernel Library jest biblioteką numeryczną firmy Intel zawierającą procedury algebry liniowej (funkcje BLAS poziomu 1, 2 i 3), Linear Algebra Package (LAPACK), Scalable Linear Algebra Package (ScaLAPACK), szybkiej transformacji Fouriera (FFT) i operacji wektorowych. Biblioteka MKL została zoptymalizowana dla procesorów Intel (może być również wykorzystywana na procesorach firmy AMD) i jest przystosowana do wielowątkowego przetwarzania równoległego w systemach wielordzeniowych, w tym również w koprocesorach Intel® Xeon Phi™.

Biblioteka MKL może być wykorzystana na koprocesorach Intel® Xeon Phi™ na trzy sposoby:

- Tryb natywny MKL - funkcje biblioteki MKL są wywoływane z programu uruchomionego bezpośrednio na koprocesorze (w trybie natywnym) lub jako niezależny węzeł obliczeniowy przy wykorzystaniu biblioteki MPI.
- Automatyczny tryb offload MKL - obliczenia realizowane przez wybrane funkcje biblioteki MKL są przez kompilator automatycznie rozdzielane do przetwarzania zarówno przez CPU jak i koprocesor. Kompilator również zapewnia automatyczne przesyłanie danych do/z koprocesora niezbędnych do realizacji obliczeń.
- Tryb offload MKL wspierany przez kompilator - funkcje z biblioteki MKL wywoływane są z wnętrza fragmentów kodu realizowanych na koprocesorze (w trybie offload). W tym trybie do zarządzania przesyłem danych do i z koprocesora są wykorzystywane pragmy i dyrektywy kompilatora.

9.1 Tryb natywny MKL

Przykładowy program, wykorzystujący bibliotekę MKL do mnożenia macierzy przedstawiono poniżej (nazwa pliku `mkl_native.c`):

```
#include <mkl.h>

#define n 20000

int main()
{
    double *A, *B, *C;
    A = (double *)mkl_malloc(n*n*sizeof(double), 128);
    B = (double *)mkl_malloc(n*n*sizeof(double), 128);
    C = (double *)mkl_malloc(n*n*sizeof(double), 128);

    double time_start = dsecnd();
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                n, n, n, 1.0, A, n, B, n, 0.0, C, n);
    double time_end = dsecnd();
    printf("Running time: %fs\n", time_end - time_start);

    mkl_free(A);
    mkl_free(B);
    mkl_free(C);
}
```

W powyższym kodzie jest wykonywana operacja $\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B}$ o rozmiarach $n \times n$ przy wykorzystaniu funkcji `dgemm` z biblioteki MKL. Dla uproszczenia w powyższym programie pominięto inicjalizację macierzy oraz obsługę błędów.

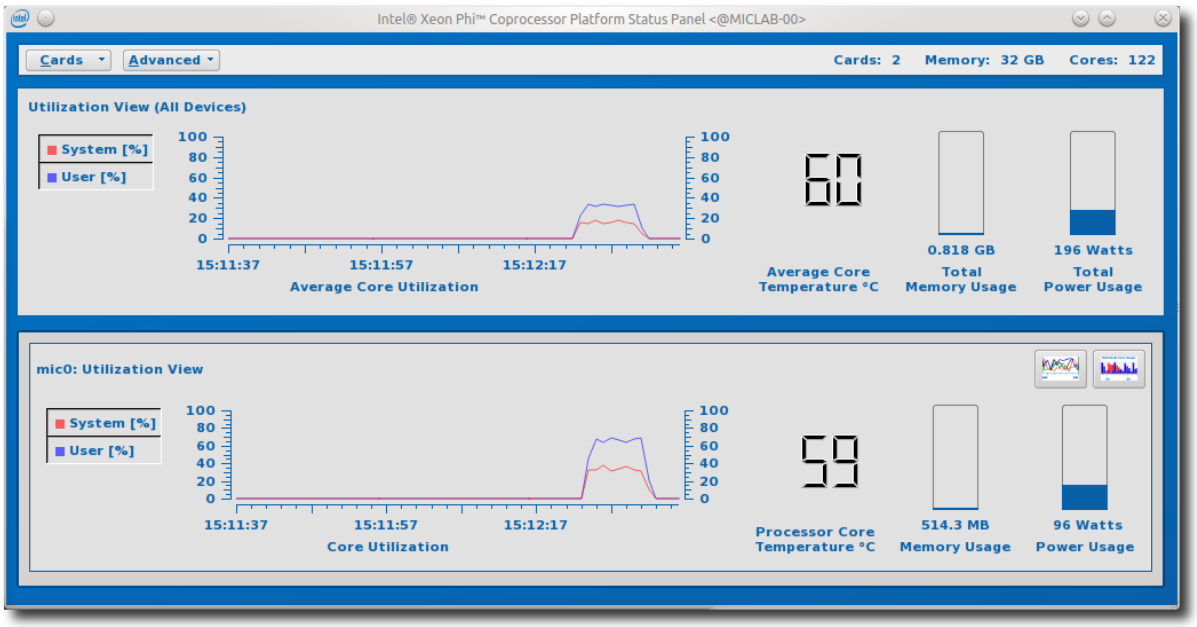
Dla tego trybu kompilacje należy przeprowadzić na hoście i użyć oprócz opcji kompilatora `-mmic` również opcje `-mkl` (zazwyczaj zachodzi konieczność dodania ścieżek biblioteki MKL do ścieżek przeszukiwań plików nagłówkowych kompilatora, jednakże na maszynach udostępnionych w ramach projektu MICLAB nie trzeba tego robić). Przykładowa kompilacja i uruchomienie programu może wyglądać następująco:

```
[user@MICLAB-01 ~]$ icc -mmic -mkl mkl_native.c
[user@MICLAB-01 ~]$ micnativeloadex ./a.out
```

```
Running time:  51.163557s
```

Do pomiaru czasu działania programu użyto polecenia `time`. W powyższym przykładzie całkowity czas wykonania programu z wykorzystaniem jednego koprocesora wyniósł 8,292 sekundy.

Poniżej znajduje się przykładowy zrzut ekranu narzędzia *Intel® Xeon Phi™ Coprocessor Platform Status Panel* (polecenie `micsmc`), który ilustruje wykorzystanie koprocesora `mic0` w trakcie wykonywania tego programu:



9.2 Automatyczny tryb offload MKL

Aby skorzystać z automatycznego trybu offload MKL należy ten tryb włączyć. Można to zrobić na dwa sposoby. Pierwszy sposób polega na wykorzystaniu funkcji biblioteki MKL, które umieszczamy w kodzie źródłowym. Do włączenia automatycznego rozdzielania zadań służy funkcja `mkl_mic_enable` (wyłączenie tego trybu jest realizowane poprzez funkcję `mkl_mic_disable`). Druga metoda wykorzystuje zmienne środowiskowe. Przed uruchomieniem programu należy zmienną środowiskową `MKL_MIC_ENABLE` ustawić na wartość `1`.

Po włączeniu tego trybu wewnętrzne mechanizmy biblioteki MKL rozdzielą zadania zarówno dla procesora (lub procesorów), jak i koprocesorów. Niezależnie od tego, który sposób zostanie wybrany, kod programu należy skompilować dla CPU.

Przykładowe uruchomienie programu z wcześniejszego podpunktu w tym trybie może wyglądać następująco:

```
[user@MICLAB-01 ~]$ icc -mkl mkl_native.c
[user@MICLAB-01 ~]$ export MKL_MIC_ENABLE=1
[user@MICLAB-01 ~]$ ./a.out

Running time:  15.662728s
```