3.Opisać cechy, znaczenie i rolę w masywnych obliczeniach równoległych: Optymalizacji programów uwzględniającej różne aspekty architektur.

Jest kilka możliwości, które pozwalają zwiększyć szybkość działania programu. Są to metody, począwszy od zastosowania odpowiedniego kompilatora i jego przełączników. Takie postępowanie nie wymusza pisania kodu programu od początku, czy też jakąkolwiek zmianę strukturalną kodu wykonaną przez programistę. Następne metody polegają na zmianie struktury kodu, w celu wykorzystania największej ilości mocy obliczeniowej, która może być wykorzystana przez pracujący program. Takimi technikami są np. wątki, OpenMP oraz MPI.

Metody optymalizacji kodu to między innymi:

**- Wyliczanie wartości stałych**

**-Upraszczanie wyrażeń logicznych i arytmetycznych**

**-Eliminacja zbędnych wyrażeń**

**-Łączenie pętli**

**-Spłaszczanie pętli**

**-Eliminacja identycznych argumentów funkcji**

**-Eliminacja nieosiągalnego kodu**

**-Eliminacja zbędnych instrukcji warunkowych**

- itp.

Optymalizacja na przykładzie kompilatora GCC:

Kompilator [GCC](http://pl.wikipedia.org/wiki/GNU_Compiler_Collection) przeprowadza najpierw optymalizację niezależną od architektury, a następnie,jeśli użytkownik sobie tego zażyczy, optymalizację kodu pod konkretny procesor lub nawet model procesora.

Optymalizacja ogólna jest wykonywana na zasadach, które są wspólne dla wszystkich maszyn, architektur i procesorów. Użytkownik może samodzielnie wybrać optymalizacje, które mają zostać wykonane, albo wybrać jeden ze zdefiniowanych poziomów optymalizacji (przełączniki -O, -O2, -O3, -Os):

* -O – optymalizacja tylko w zakresie podstawowym, skraca to proces kompilacji programu;
* -O2 – wszystkie bezpieczne optymalizacje, tzn. nie zostanie włączona żadna optymalizacja, która mogłaby zmienić w istotny sposób działanie programu (np. zmniejszyć precyzję obliczeń zmiennoprzecinkowych);
* -O3 – najwyższy poziom optymalizacji, może powodować problemy w działaniu skomplikowanych programów, zmniejszyć precyzję obliczeń i spowodować znaczny wzrost objętości kodu maszynowego programu;
* -Os – kod optymalizowany w celu minimalizacji rozmiaru pliku wykonywalnego;

Wbrew pozorom najwyższy poziom optymalizacji (-O3) nie musi przekładać się na szybsze działanie programu. Dłuższy kod wynikowy sprawia, że stosunkowo niewielka jego część da się zmieścić się w szybkiej pamięci podręcznej procesora.

Kompilator może również wykonać wiele optymalizacji, aby program lepiej działał na konkretnej maszynie. Jest to związane z pewnymi charakterystycznymi cechami procesora, takimi jak: ilość i rodzaje rejestrów (np. Procesory PowerPC mają więcej rejestrów ogólnego przeznaczenia niż Pentium), ilość i algorytmy zarządzające pamięcia cache (aby zmaksymalizować jej użycie). Rodzaje instrukcji (z naciskiem na instrukcje wyspecjalizowane do obróbki pewnych typów danych oraz instrukcje SIMD), wielopotokowość, czy superskalarność mają również ogromne znaczenie z punktu widzenia optymalizacji.

GCC na platformie x86 potrafi optymalizować kod pod szereg różnych procesorów. Włączenie optymalizacji powoduje często, że kod wynikowy nie uruchomi się na innym, podobnym procesorze lub będzie na nim działał wyjątkowo wolno.

Optymalizacja przy użyciu wielowątkowości

**Wielowątkowość** – cecha systemu operacyjnego, dzięki której w ramach jednego procesu może wykonywać kilka zadań lub jednostek wykonawczych. Nowe zadania to kolejne ciągi instrukcji wykonywane oddzielnie. Wszystkie zadania w ramach tego samego procesu współdzielą kod programu i dane.

**Wielowątkowość** może także odnosić się do samych procesorów. W takim wypadku oznacza możliwość jednoczesnego wykonywania wielu wątków sprzętowych na pojedynczej jednostce wykonawczej – rdzeniu.

**OMP**

wieloplatformowy interfejs programowania aplikacji (API) umożliwiający tworzenie programów komputerowych dla systemów wieloprocesorowych z pamięcią dzieloną. Może być wykorzystywany w językach programowania C, C++ i Fortan na wielu architekturach, m.in. Unix i Microsoft Windows. Składa się ze zbioru dyrektyw kompilatora, bibliotek oraz zmiennych środowiskowych mających wpływ na sposób wykonywania się programu.

Celem OpenMP jest implementacja wielowątkowości, czyli metody zrównoleglania programów komputerowych, w której główny wątek (czyli ciąg następujących po sobie instrukcji) „rozgałęzia” się na kilka „wątków potomnych”, które wspólnie wykonują określone zadanie. Wątki pracują współbieżnie i mogą zostać przydzielone przez środowisko uruchomieniowe różnym procesorom.

**MPI**

protokół komunikacyjny będący standardem przesyłania komunikatów pomiędzy procesami programów równoległych działających na jednym lub więcej komputerach. Interfejs ten wraz z protokołem oraz semantyką specyfikuje, jak jego elementy winny się zachowywać w dowolnej implementacji. Celami MPI są wysoka jakość, skalowalność oraz przenośność. Standard MPI implementowany jest najczęściej w postaci bibliotek, z których można korzystać w programach tworzonych w różnych językach programowania, np. C, C++, Ada, Fortan.

Zaletami MPI nad starszymi bibliotekami przekazywania wiadomości są przenośność oraz prędkość. Przenośność, ponieważ MPI został zaimplementowany dla każdej architektury opartej na rozproszonej pamięci. Prędkość, ponieważ każda implementacja jest zoptymalizowana pod sprzęt, na którym działa.