Programowanie współbieżne... (10)

Andrzej Baran **2010/11**

LINK: http://kft.umcs.lublin.pl/baran/prir/index.html

Biblioteki

- Biblioteki podstawowe
  - BLACS (**B**asic **L**inear **A**lgebra **C**ommunication **S**ubprograms)
  - BLAS (**B**asic **L**inear **A**lgebra **S**ubprograms), ATLAS (patrz sieć)
- Biblioteki złożone
  - LAPACK (**L**inear **A**lgebra **PACK**age)
  - ScaLAPACK (**Sca**lable **L**inear **A**lgebra **PACK**age)
- Inne
  - MKL
  - LINPACK (**Lin**ear **PACK**age)
  - IMSL (matematyka + statystyka)
  - gsl – **G**nu **S**cientific **L**ibrary (patrz sieć)
- Sposoby używania bibliotek

Biblioteki sekwencyjne

- BLAS zawiera podstawowe procedury operacji na wektorach i macierzach (tablicach)
  - Mnożenie przez liczbę, iloczyn skalarny, kopiowanie
  - Dodawanie wektorów, macierzy
  - Kopiowanie z mnożeniem i dodawaniem
  - itp
- BLAS posiada 3 poziomy złożoności
  - Mnożenia wektor op wektor – BLAS1 (op = operator)
  - Mnożenia macierz op wektor – BLAS2
  - Mnożenia macierz op macierz – BLAS3
- Wszystkie operacje sa zoptymalizowane dla danej architektury procesorów (patrz też GOBLAS)

Biblioteki sekwencyjne

- Adres internetowy źródeł biblioteki BLAS: http://www.netlib.org/blas/
  - Ze źródeł można utworzyć kod dla dowolnej platformy; nie jest on optymalny
- Najlepsze implementacje: gotowe kompilaty przygotwane na konkretne platformy, np. biblioteka MKL Intel Fortran (ifort)

Procedury BLAS

- Biblioteka BLAS posiada trzy poziomy. Poziom 1 biblioteki BLAS zawiera procedury, których czas wykonania jest $O(n)$ (czytaj: rzędu n). Kolejne dwa poziomy 2 i 3 zawierają procedury czasu $O(n^2)$ i $O(n^3)$
- Nazwy procedur są skrótami wyrażeń postaci „*scalar alpha x plus y*” (*iloczyn, produkt skalara alpha przez wektor x plus wektor y*). W tym przypadku nazwa procedury (funkcji) jest: **saxpy**.
- Inny sposób nazywania procedur to nazwy znanych operacji, np. **dot(x,y)** oznacza iloczyn skalarny wektorów (*dot product*) **x** i **y**.

Procedury BLAS1

- Poziom 1. Operacje na wektorach. Przykłady działań:
  - c=dot(x, y) (c = c + x(i)y(i), i=1, 2, ...)
  - z=saxpy(alpha, x, y) (z(i) = alpha*x(i) + y(i), i=1,....n)
- W zależności od typu argumentów, przed nazwą procedury może wystąpić dodatkowa litera d (od double precission), c (od complex) np. ddot(), cdot()
- Łatwo sprawdzić, że szybkość wykonywania operacji BLAS jest o wiele większa niż szybkość realizacji wewnętrznych (intrinsic) operacji macierzowych dostępnych w FORTRAN 95 (nie mówiąc już o własnych realizacjach takich działań).
- Zadanie. Sprawdź praktycznie powyższe stwierdzenie.

Procedury BLAS2, 3

- Poziom 2: $O(n^2)$ Mnożenie macierz x wektor
  - z = y + A x; z = gaxpy(A, x, y) (Tutaj: dim(A) = m x n, dim(x) = n, dim(y) = m)
  - Nazwa gaxpy pochodzi od wyrażenia „general A x plus y”
- Poziom 3: $O(n^3)$ Mnożenia typu macierz x macierz (C = A B)
  - Mnożenie można zrealizować na sześć różnych sposobów używając procedur niższych poziomów (Zadanie: wykonaj te mnożenia) Odpowiadający poziom to BLAS3
- Zarówno dokładny jak i skrótowy opis procedur BLAS w wersji FORTRAN jak i C, C++ można znaleźć w internecie

LAPACK

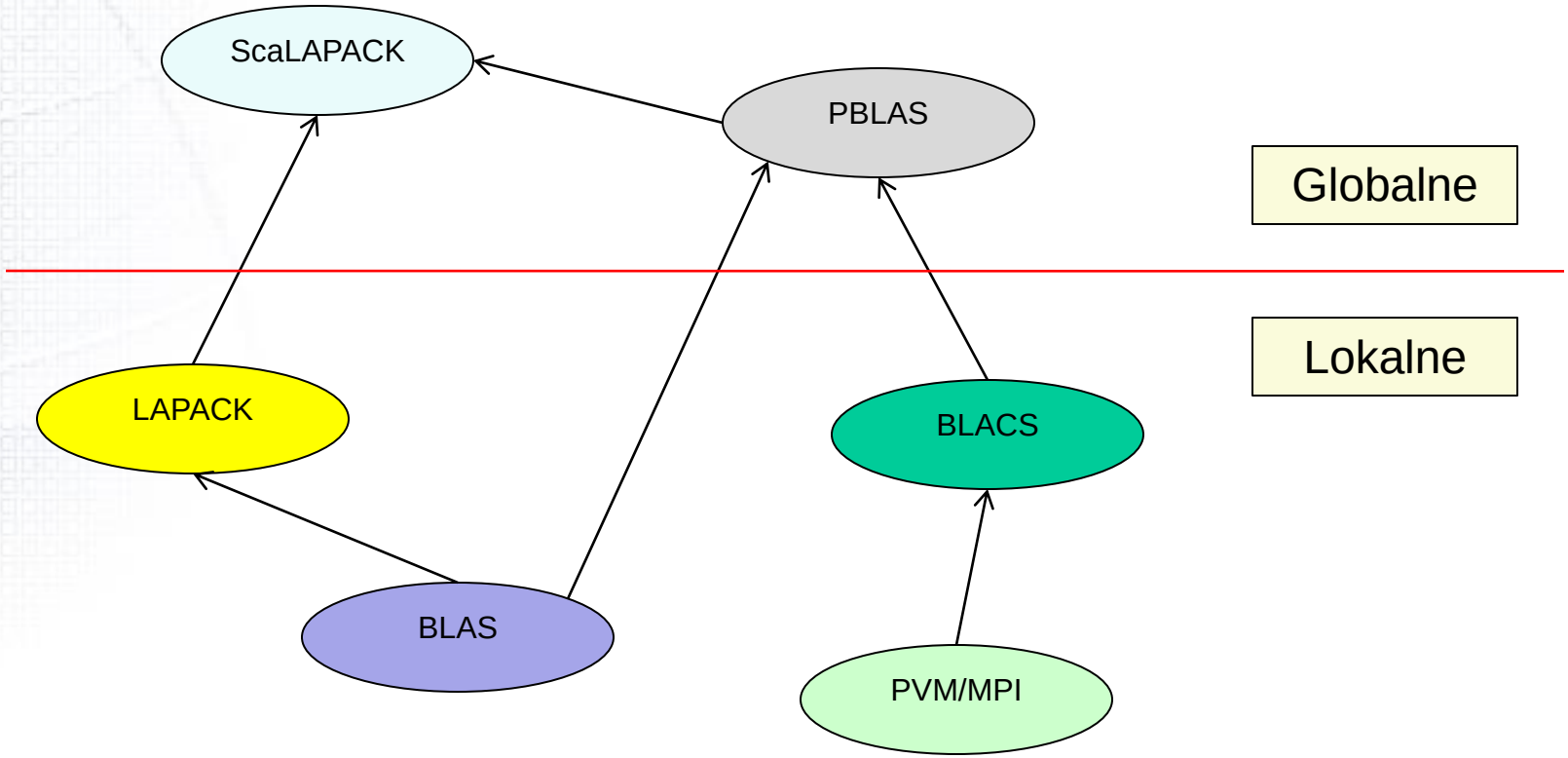
- Rozwiązywanie układów równań liniowych
- Wartości własne i wektory własne macierzy
- Linear least-square fitting

BLACKS

- Podstawowe operacje
  - związane z przekazywaniem danych między procesami
  - Organizacja sieci procesów (gridy)

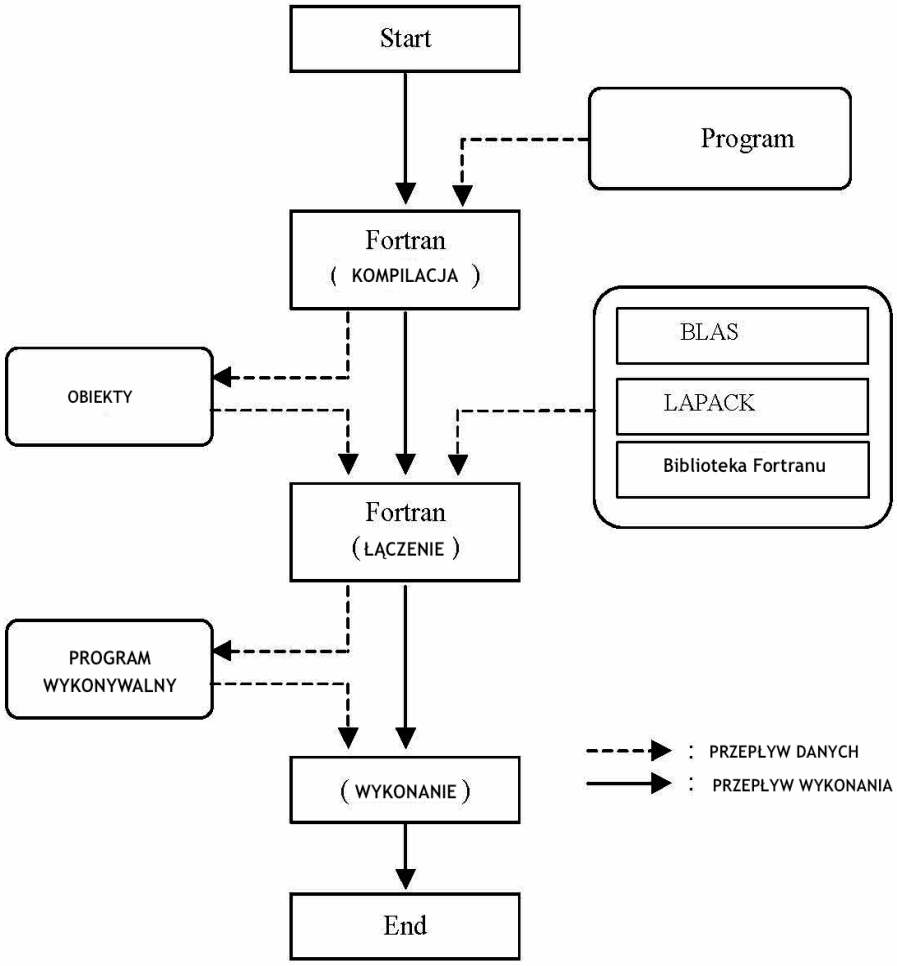
Biblioteki

Hierarchia bibliotek matematycznych



Kompilacja z BLAS, LAPACK

- Kompilacja programów z użyciem bibliotek zewnętrznych (spec)
- Dokładne omówienie na ćwiczeniach (Lab)
- W czasie łączenia obiektów w wykonywalny program linker () dołącza procedury biblioteczne



Co robi make?

- Program **make** wykonuje program zapisany w pliku **Makefile**
- **Makefile** zawiera polecenia dla powłoki – systemu.
- Polecenia mogą dotyczyć wieloetapowego procesu kompilacji programu (wiele plików, modułów, procedur)
- Długie polecenia kontynuujemy w następnej linii po znaku \ (tylko on)
- Polecenia wpisujemy po znaku <tab> (wcięte linie)
- Linie ze znakiem : podają zależności między różnymi plikami (np. %.o:%f oznacza, że pliki z rozszerzeniem .o zależą of plików z rozszerzeniem .f)
- make clean – usuwa zbędne pliki

```
# Plik makefile/Makefile
# Definicje zmiennych
OBJECT = global.o main.o funct.o subr.o
COMP = ifort    # kompilator
OPT = -O3       # opcje kompilacji
# Makefile
exe: $(objects)
        $(COMP) -o exe $(OPT) $(OBJECT)
global.mod: global.o global.f90
        $(COMP) -c $(OPT) global.f90
global.o: global.f90
        $(COMP) -c $(OPT) global.f90
main.o: global.mod main.f90
        $(COMP) -c $(OPT) main.f90
funct.o: global.mod funct.f90
        $(COMP) -c $(OPT) funct.f90
%.o: %.f90
        $(COMP) -c $(subr) $<
# Czyszczenie
clean:
        rm global.mod
        rm $(OBJECT)
# Koniec makefile
```

Zyski z make

- Makefile tworzymy raz!
- Aby wykonac wszystkie polecenia z Makefile piszemy tylko make lub make etykieta
- Po poprawkach dokonanych w dowolnym pliku wydajemy polecenie make co powoduje kompilację tego tylko pliku oraz powoduje wykonanie tych poleceń z makefile, które od tego pliku zależą
- Skraca się przez to czas pracy i nie musimy za każdym razem mozolnie wystukiwać na klawiaturze wszystkich potrzebnych poleceń
- Wygodnie jest mieć w danym katalogu programy z jednego projektu i tylko jeden makefile

Zyski z make

- Jeśli używamy bibliotek definiujemy je w makefile
- Podbnie, definiujmy w makefile ścieżki do plików, których używamy w projekcie lub ścieżki do bibliotek
- Program make rozpoznaje sam pliki makefile lub Makefile
- Z make można robić wiele rzeczy w łatwy sposób!

- Więcej o make i Makefile znajdziesz pod adresem www: http://www.gnu.org/software/make/manual/make.html

Problemy...?