

# Implementacja solwera frontálnego zrównoleglonego w wielowężłowym heterogenicznym środowisku sprzętowym

Paweł J. Wal

Opiekun pracy: dr inż. Łukasz Rauch

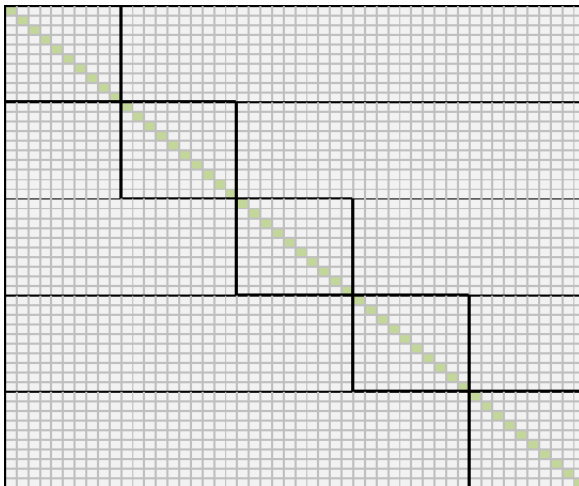
Wydział Inżynierii Metali i Informatyki Przemysłowej  
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

7 maja 2015

# Oryginalny pomysł - solver multifrontalny na GPU

- Wykorzystany OpenCL
- Masowo równoległe przetwarzanie na GPU
- Wydzielenie frontów rozwiązania
  - Podział macierzy na części (dekompozycja)
  - Wydzielenie z macierzy rzadkiej gęstych fragmentów
- Przetworzenie frontów rozwiązania
  - Przetwarzanie części niezależnie od siebie
  - Rozwiązanie zależności - na koniec

# Oryginalny pomysł - dekompozycja



# Wady oryginalnego podejścia

- Użycie tylko jednego z dostępnych urządzeń
- Uwarunkowania pamięciowe
  - Ładowanie stosunkowo dużych fragmentów macierzy do pamięci
  - Dodatkowa pamięć potrzebna na specyficzne struktury danych używane przez solver (mapowanie wierszy)
- Niewielka odporność na zdarzenia katastrofalne (np. awarię zasilania)

# Nowy kierunek

- Użycie wszystkich dostępnych urządzeń
- Wykorzystanie możliwości maszyn z wieloma GPU?
  - Wiele GPU = specjalne płyty główne i obudowy = koszt
  - Wiele GPU = specjalne systemy operacyjne i konfiguracje = maszyna o wąskim zastosowaniu
  - Potęga GPU = moc urządzeń klasy konsumenckiej
- Myśl przewodnia: realne zastosowanie w przemyśle i nauce

# Założenia projektu

- Możliwość wykorzystania dziesiątek, setek, ... urządzeń dowolnej klasy - przez OpenCL
- Dowolna lokalizacja urządzeń
- Dowolne dopuszczalne obciążenie urządzeń
- Brak nierealistycznych lub wygórowanych założeń (najlepiej: brak założeń w ogóle):
  - Ilość urządzeń w maszynie
  - Moc obliczeniowa urządzeń
  - Rodzaj i jakość połączenia (filtrowane, komórkowe)
  - Czas i wydajność pracy

# Założenia projektu

- Rozwiązywanie wielu problemów na raz
- Rozwiązywanie problemów uprzednio praktycznie nierozwiązalnych
- Odporność na błędy - od drobnych do katastrofalnych

# Inspiracja





# Schemat rozwiązania



# Przewaga rozwiązań

- Wykorzystanie ukrytego (marnowanego) potencjału urządzeń
- Niewielkie wymagania własne
  - Pracuje z dostępnymi technologiami i popularnymi urządzeniami
  - Nie wymaga inwestycji
  - Nie wymaga specjalistycznej wiedzy by być skutecznie użytkowanym
- Pozwala na rozwiązywanie problemów niepraktycznych lub niemożliwych do rozwiązania

# Architektura

- Klient-serwer
- Dlaczego nie MPI?
  - Rozłączne LANy
  - Pojedyncze maszyny w różnych lokalizacjach
  - Gorsze połączenia
- Ostateczna decyzja? Protokół HTTP!
  - Popularność
  - Typowość - raczej nie jest blokowany

# Serwer

- Ruby on Rails
  - Jednolite, standardowe API (REST)
  - Bezstanowość
    - Rozwiązanie wielu problemów na raz
    - Odporność na awarie nagle zatrzymujące serwer
  - Ta sama usługa dostarcza punkt wejścia dla klientów i dla użytkowników (interfejs graficzny)
- Interfejs użytkownika
  - Nowoczesne technologie webowe
  - Łatwość obsługi - nawet dla laika
  - W praktyce: kolejna redukcja kosztów (specjaliści, szkolenia, ...)

# Serwer - interfejs użytkownika

Swarmhost - Chromium

Swarmhost

localhost:3000/admin/solutions/new

Aplikacje MadeOnMoon sc2planner mom\_admin - Server-generator pegboard nerd » Inne zakładki

**SWARMHOST**

Logout

Nodes

**Solutions**

Tasks

## New Solution

matrix  Nie wybrano pliku

right-hand-side vector  Nie wybrano pliku

size of part

matrix dimension

Swarmhost © Paweł J. Wal. Insight Template by WebThemez

localhost:3000/admin/nodes

# Serwer - interfejs użytkownika

Swarmhost - Chromium

Swarmhost

localhost:3000/admin/solutions

Aplikacje MadeOnMoon sc2planner mom\_admin - Server-generat pegboard nerd » Inne zakładki

**SWARMHOST**

Nodes

Solutions

Tasks

## Solutions

Name	Waves	Tasks	State	Created at	Completed	Actions
e30r0100	15	75	complete	Sat, 02 May 2015 16:35:35 +0200	Sat, 02 May 2015 17:18:50 +0200	Delete
e30r0100	8	80	available	Sat, 02 May 2015 19:27:17 +0200		Delete

Swarmhost © Paweł J. Wal, Insight Template by WebThemez

# Klient

- Ruby
  - Łatwość prototypowania
  - Dynamiczny rozwój oprogramowania
- Usługa rezydentna
  - Pracuje tylko wtedy kiedy serwer podaje zadania
  - Niewielki narzut ze strony samego klienta
- Pełna asynchroniczność

# Elementy obliczeniowe

- Trudno przebić C/C++
  - Szybkość
  - Obsługa pamięci
  - Dostępność bibliotek numerycznych (np. Boost C++ uBLAS)
- Rozwiązanie:  
Ruby Native Extensions
  - Uruchomienie kodu natywnego w kontekście Ruby
  - Rozwiązanie hybrydowe





# Przewaga użytych technologii

- Łatwość rozszerzania serwera - prosty język, jasna struktura danych
- Łatwość modyfikacji klienta
- Łatwość stworzenia zupełnie nowego klienta
- Cały dynamizm Ruby, cała szybkość C++, cała moc OpenCL i GPU
- Łatwość dostosowania do aktualnie postawionego celu przez specjalistę
- Łatwość użycia jako gotowego rozwiązania przez specjalistę w innej dziedzinie

# Dalszy rozwój

- Rozwiązanie jest nadal w rozwoju
- Aktualnie badane obszary:
  - Usprawnienie komunikacji po HTTP
  - Obniżenie obciążenia serwera
  - Zwiększenie asynchroniczności kluczowych elementów
  - Badanie wpływu parametrów podziału macierzy na prędkość uzyskiwania wyników
  - Badanie wpływu parametrów uruchomieniowych OpenCL na szybkość uzyskiwania rozwiązań na kliencie
  - I inne.