SLAJD 1

Dzień dobry, nazywam się Paweł Wal. Zaprezentuję projekt inżynierski dotyczący realizacji frontalnego solwera metody elementów skończonych z wykorzystaniem technologii OpenCL, którego promotorem był doktor Łukasz Rauch.

SLAJD 2

Oryginalny solwer frontalny zaproponował Bruce Irons w 1970 roku. Motywacją dla jego pracy była relatywnie niska moc obliczeniowa ówczesnych procesorów, ograniczona pamięć operacyjna – pracował na urządzeniu ICT 1905 które dysponowało tylko 96kB – oraz rosnący rozmiar problemów wymagających rozwiązania układów równań liniowych. Z podobnymi problemami mierzymy się na GPU – w stosunku do hostów pamięć operacyjna jest ograniczona, a jej wypełnianie – czasowo kosztowne.

SLAJD 3

Motywacją projektu było wykorzystanie idei solwera frontalnego według Ironsa i – budując na jego oryginalnych pomysłach – stworzenie autorskiego rozwiązania wykorzystującego możliwości równoległości masowej dostarczanej przez platformy GPGPU. Celami, bądź założeniami przyświecającymi projektowi był paradygmat czarnej skrzynki – praca bez przyjmowania założeń wobec oprogramowania z którym będzie integrowany solwer. Dzięki temu zaproponowane rozwiązanie jest uniwersalne – może rozwiązywać dowolne problemy wyrażone układami równań liniowych – oraz jest łatwe w implementacji. Założyłem również przenośność między systemami operacyjnymi i urządzeniami obliczeniowymi.

SLAJD 4

Oto zastosowane podejście. Przede wszystkim w oryginalnej macierzy wydzielane są części, czyli fronty rozwiązania. Na slajdzie przedstawiony jest podział na części dla macierzy diagonalnej -jest to idealny przypadek, użyty dla jasności. Z oryginalnej macierzy wydzielane są fragmenty które nie muszą być symetryczne, o rozmiarach zależnych od wysuniętych najbardziej na lewo oraz na prawo wyrazów niezerowych w wierszach należących do części.

SLAJD 5

Zastosowano wariant metody Gaussa który pozwala na równoległą implementację – przypisując jeden wątek do każdego wiersza macierzy. W metodzie Gaussa wykorzystywane są operacje elementarne – mnożenie i dodawanie wierszy, oraz – celem uzyskania macierzy schodkowej – zamiana wierszy.

By uniknąć kosztownej pamięciowo zamiany wierszy, wykorzystałem dodatkowy wektor, stanowiący mapę pozycji faktycznej wiersza do pożądanej pozycji w macierzy schodkowej. Pożądana pozycja to oczywiście pierwszy wyraz niezerowy w wierszu, patrząc od lewej. Zastosowanie mapy pozwala na szybką weryfikację unikalności, oraz w przypadku w którym tej unikalności brak – szybko daje informację względem którego wiersza należy zredukować ten aktualnie przetwarzany.

Na slajdzie widać kilka pierwszych operacji podczas działania tego algorytmu na przykładowej macierzy. Wątki przetwarzające wiersze – w tym przypadku wiersze r1, r3 i r4 – sprawdzają pierwszy wyraz niezerowy, znajdują odpowiadające im pozycje w mapie nieobsadzone, i wpisują tam swoje indeksy.

SLAJD 6

Na tym slajdzie przedstawiono przykład wpisania z eliminacją. Wątek obsługujący wiersz r5 znajduje informację że pozycja x3 jest już obsadzona, dokonuje eliminacji pierwszego wyrazu względem wiersza r3 i obsadza pozycję x4.

SLAJD 7

Podobną sytuację przedstawiono tutaj, dla wiersza r2. W jego przypadku, by dojść do nieobsadzonej w mapie pozycji, konieczne są trzy eliminacje. Warto zauważyć, że po wykonaniu tych operacji można już przeprowadzić – sekwencyjnie - podstawienie wstecz, wykorzystując mapę do wyboru wierszy dla kolejnych funkcji.

SLAJD 8

Dla równoległości masowej algorytm wymagał drobnych dostosowań. Przede wszystkim konieczne było ujęcie koncepcji grup roboczych, wydzielonych w obrębie części macierzy.. Proponowany algorytm przywraca formę macierzy schodkowej w obrębie jednej grupy roboczej. By przywrócić ją w obrębie całej części macierzy, zastosowano dodatkowy kernel na GPU. Dla całej macierzy analogiczna do jego działania jest przeprowadzana na CPU.

SLAJD 9

Na tym slajdzie przedstawiono przykład przetwarzania jednej części macierzy na GPU. Dla każdej grupy funkcjonuje osobna, lokalna mapa. Macierz na slajdzie jest pokazana już po zastosowaniu algorytmu w obrębie grup roboczych; wewnątrz nich przywrócono formę macierzy schodkowej. W obrębie frontu jednak warunek nie jest spełniony.

SLAJD 10

Drugi kernel identyfikuje konfliktujące w obrębie frontu wiersze i prowadzi ich eliminację. Jeśli kernel ten zasygnalizuje że wprowadził zmiany, ponownie wykonywany jest kernel pierwszy, by upewnić się że forma macierzy schodkowej zostanie przywrócona w obrębie grup roboczych.

SLAJD 11

Wykonanie kerneli powtarzane jest dopóki drugi z nich nie zgłosi zerowej ilości wykonanych operacji. Wtedy zachodzi pewność że w obrębie części macierzy przywrócona jest forma macierzy schodkowej.

SLAJD 12

Przeprowadzone zostały też badania wydajności. Przede wszystkim ustalono dla dwóch urządzeń optymalną lokalną ilość wątków, czyli ilość wątków na grupę roboczą. Dla karty Tesla M2090 najlepsze wyniki uzyskano dla 256 wątków na grupę…

SLAJD 13

…zaś dla procesorów Intel Xeon X5650 – 128.

SLAJD 14

Następnie w oparciu o te dane przeprowadzono badania ile wątków powinno przypadać na jedną część macierzy, czyli na front. Dla karty Tesla najlepsze wyniki uzyskano przy 768 i 1024 wątkach na front, czyli przy ilości trzy- oraz czterokrotnie większej w stosunku do lokalnej. Nie są to jednak duże przyspieszenia.

SLAJD 15

Dla procesora Intel Xeon najlepiej wypadła globalna ilość wątków równa 512, czyli również czterokrotność ilości wątków na grupę roboczą w obrębie frontu. Dla procesora Xeon przyspieszenie jest lepiej widoczne.

SLAJD 16

Przeprowadzono również badania sprawdzające zużycie czasu na wiersz macierzy przy optymalnej lokalnej ilości wątków i rosnącej ilości globalnej. Niebieskim kolorem zaznaczono czas na element macierzy dla macierzy o stałym rozmiarze, a zielonym – dla macierzy proporcjonalnie większej (czyli o rozmiarze równym rozmiarowi wyjściowej macierzy razy stosunek ilości lokalnej do globalnej ilości wątków, odpowiednio dwa, cztery, osiem i szesnaście razy większej.

Tutaj pokazano wyniki dla karty Tesla. Czas na element pozostaje podobny wraz ze wzrostem problemu, co oznacza że rozwiązanie jest dość dobrze skalowalne.

SLAJD 17

Tutaj uwidoczniono podobne wyniki dla procesora Intel Xeon. Jak widać, dla proporcjonalnie większej macierzy niekiedy czas per element jest nawet lepszy dla większej macierzy, jak widać przy 1024 wątkach globalnych.

SLAJD 18

Podsumowując, w ramach projektu został stworzony możliwy do zrównoleglenia wariant eliminacji Gaussa. Został zaproponowany i zaimplementowany frontalny, masowo równoległy solwer wykorzystujący OpenCL, stworzony zgodnie z paradygmatem czarnej skrzynki. Przeprowadzono badania proponowanego rozwiązania i sformułowano dotyczące go wnioski, identyfikując silne i słabe strony stworzonego oprogramowania.