Prezentacja sesja kół - shownotes

Slajd 1

Tytułowy

Slajd 2

Podczas poprzedniej Sesji Kół Naukowych miałem zaszczyt prezentować mój oryginalny pomysł - solwer multifrontalny na GPU. Był on stworzony przy użyciu frameworka OpenCL i wykorzystywał masowo równoległe przetwrzanie na GPU.

Slajd 3

Jednym z ciekawszych elementów tego pomysłu była metoda dekompozycji macierzy - z macierzy rzadkiej były wydzielane części, obiektywnie patrząc, gęste, do przetwarzania których był przystosowany solwer. Części mogły być przetwarzane w dużej mierze niezależnie od siebie - tylko pewne ich obszary były od siebie wzajemnie zależne. Przetworzenie tych zależności można było zrealizować już po współbieżnym przetworzeniu części.

Slajd 4 (wady oryginalnego podejścia)

Jak każde, zaproponowane wtedy rozwiązanie miało pewne wady. Przede wszystkim, wykorzystywało tylko jedno z dostępnych urządzeń OpenCL; już wtedy, dzięki uprzejmości Katedry Informatyki Stosowanej i Modelowania, miałem do dyspozycji maszynę z czterema procesorami graficznymi, więc był to pierwszy podjęty kierunek rozwoju. Mimo zastosowania sprytnych optymalizacji, solwer wymagał też sporo pamięci, a w przypadku awarii systemu operacyjnego lub braku zasilania rozwiązanie trzeba było powtórzyć.

Slajd 5 (nowy kierunek)

Dlatego zidentyfikowałem nowy kierunek moich badań. Przede wszystkim, chciałem użyć wszystkich dostępnych urządzeń - ale zacząłem myśleć o tym, jak zbudowane są takie "naukowe" maszyny jak ta z którą pracuję. Owszem, ma wiele GPU - bardzo mocnych i bardzo drogich, na specjalnej płycie głównej, również kosztującej więcej niż konsumencka, umieszczonej w specjalistycznej obudowie. Cała konfiguracja systemu operacyjnego jest specyficzna, co powoduje że maszyna ta ma wąskie zastosowania. A cała potęga GPU - podkreślam to przy każdej okazji - tkwi w zastosowaniu urządzeń klasy konsumenckiej. Moją główną myślą było stworzenie oprogramowania które to wykorzysta i będzie miało praktyczne zastosowania.

Slajd 6 (założenia projektu)

Głównym założeniem projektu było wykorzystanie każdego sprzętu do jakiego uda się uzyskać dostęp, a który wspiera OpenCL. Nieważne gdzie ten sprzęt jest fizycznie umieszczony, co jeszcze takie urządzenie "robi", nieważne jaką ma moc, jakie ma połączenie z internetem i jak długo oraz jak wydajnie może pracować. Nic nie może się zmarnować. Oprócz tego nie chciałem już sytuacji, w której wrzucam do solwera jeden układ równań i czekam na jego rozwiązanie zanim wrzucę kolejny - chciałem rozwiązywać wiele problemów na raz. Skoro już usuwam uwarunkowania pamięciowe, to chciałem też zająć się naprawdę dużymi układami równań - nie takimi z tysiącem niewiadomych, ale z dziesiątkami lub setkami tysięcy niewiadomych, jeśli nie większymi. Oprogramowanie miało też być naturalnie odporne na błędy: od błędów formatu pliku, do całkowitej katastrofy.

Slajd 7 (inspiracja)

Zainspirowały mnie istniejące już projekty przetwarzania rozproszonego na masową skalę, takie jak SETI@home - na podstawie którego później stworzono Berkeley Open Infrastructure for Network Computing, BOINC, czy Folding@home. Patząc na schemat działania tych usług, zobaczyłem potencjał wykorzystania tu ponownie mojej metody dekompozycji macierzy na niewielkie części; takie części mogłyby stanowić osobne zadania, a serwer mógłby je tylko podawać pewnej, właściwie dowolnej ilości klientów.

Slajd 8 (schemat)

Schemat rozwiązania jest więc bardzo prosty: mając macierz, podajemy ją serwerowi. Serwer stosuje mój algorytm dekompozycji - dzieli macierz na części, na osobne zadania. Potem rozdziela te zadania według możliwości i zapotrzebowania między podłączonych do niego klientów; nazywam to falą rozwiązania. Kiedy fala zostaje zakończona, serwer sprawdza - według zależności między częściami - czy osiągnięto już rozwiązanie, czy konieczna jest kolejna fala rozwiązania, i zachowuje się odpowiednio.

Slajd 9 (przewaga rozwiązania)

Wspomniane projekty wykorzystują zmarnowany potencjał urządzeń, które mają zasilanie, ale nie wykonują pracy. Uznałem to za dobry kierunek i wzór do naśladowania. Stworzyłem oprogramowanie podobne do BOINC pracujące z dostępnymi technologiami i popularnymi urządzeniami bez dodatkowych inwestycji, ale jeszcze prostsze - żeby mogli z nim pracować nie tylko programiści. Zrealizowałem też swój - powiem nieskromnie - ambitny cel: stworzenie możliwości

rozwiązywania problemów niepraktycznych lub niemożliwych do rozwiązania ze względu na rozmiar.

Slajd 10 (architektura)

Podobnie jak w BOINC, wybrałem architekturę klient-serwer. Z początku rozważałem użycie MPI, de facto standardu w programowaniu rozproszonym, ale ostatecznie zrezygnowałem; pomyślałem o przypadkach gdzie mamy maszyny w rozłącznych sieciach i lokalizacjach, bądź na gorszych połączeniach. Wybrałem więc protokół HTTP: jest popularny, co ułatwi tworzenie klienta, i nie jest blokowany, co umożliwia współpracę przez filtrowane połączenia.

Slajd 11 (oprogramowanie)

Serwer został zbudowany w Ruby on Rails. Zapewnia to jednolite API zewnętrzne, bezstanowość serwera - dzięki której uzyskuję odporność na błędy i możliwość wielu rozwiązań na raz - oraz w ten sposób ta sama usługa zapewnia i interfejs użytkownika, i komunikację maszyna-maszyna. Elementy obliczeniowe są napisane w C i C++ - jak wiadomo, prędkości i zarządzania pamięcią w tym języku nic nie przebije.

Slajd 12 (przede wszystkim łatwość)

W skutek wyboru technolgii - uzyskujemy łatwość. Łatwo będzie rozszerzać serwer, łatwo zmodyfikować klienta lub stworzyć zupełnie nowego, traktując moje rozwiązanie jako framework. Programista może bardzo głęboko zmodyfikować oprogramowanie i dostosować je do swoich potrzeb, zaś specjalista w innej dziedzinie może po prostu zacząć używać rozwiązania.

Slajd 13 (serwer UI)

Aby zlecić rozwiązanie zadania, trzeba wypełnić trzy pola i nacisnąć przycisk w przeglądarce. Prawda że proste?

Slajd 14 (elementy obliczeniowe)

Tutaj uwidocznione są wyniki wstępnych, przykładowych badań. Macierz opisującą układ równań o 10k niewiadomych, z rozmiarem części wynoszącym odpowiednio 2048, 1024 i 512 wierszy, przetwarzałem moim oprogramowaniem podłączając do roju dwie maszyny na zmianę lub razem. Od lewej - czas rozwiązania używając tylko maszyny Jennah, mającej jeden procesor graficzny

nVidia GeForce GT630, czas z użyciem Jacka - cztery karty nVidia Tesla M2090 i procesor Intel Xeon - i czas z użyciem obu maszyn. Jak widać, ten ostatni zawsze jest najmniejszy, co udowadnia zasadność podejścia. Warto też zauważyć, że rolę serwera podczas badań spełniał laptop klasy konsumenckiej.

Slajd 15 (Dalszy rozwój)

Rozwiązanie jest oczywiście nadal rozwijane. W tej chwili badam możliwości usprawnienia komunikacji po HTTP, zwiększenia asynchroniczności i w konsekwencji zmniejszenia obciążenia serwera. Badam też wpływ parametrów podziału macierzy na uzyskiwanie wyników, oraz wpływ parametrów uruchomieniowych OpenCL na szybkość przetwarzania po stronie klienta.