Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki
Katedra Informatyki

Autoreferat rozprawy doktorskiej

Implementacje algorytmów oddziałujących cząstek na architekturach masywnie równoległych

mgr inż. Tomasz Rożen

Promotor:
dr hab. Krzysztof Boryczko prof. AGH, Akademia Górniczo-Hutnicza w Krakowie, EAIiE

Recenzenci:
dr hab. inż. Marek Tudruj prof. PAN, Polska Akademia Nauk w Warszawie, IPI
prof. dr hab. inż. Jacek Kitowski, Akademia Górniczo-Hutnicza w Krakowie, EAIiE

Cel i zakres pracy

Metody oddziałujących cząstek należą do popularnych metod symulacyjnych, umożliwiających przeprowadzanie zaawansowanych eksperymentów komputerowych, między innymi z takich dziedzin nauki jak fizyka, chemia, astrofizyka. Do przeprowadzenia sensownej symulacji popularnymi metodami cząstek, takimi jak dyssypatywna dynamika cząstek (DPD) czy hydrodynamika cząstek wygładzonych (SPH), nierzadko stosuje się układy zbudowane z milionów cząstek lub większe. Pociąga to za sobą duże nakłady obliczeniowe, a szukanie efektywnych algorytmów symulacji jest ciągle przedmiotem badań. W pracy doktorskiej rozpatrywane były jedynie metody cząstek o oddziaływaniach krótkozasięgowych, do których należą m.in. wymienione metody.

W ciągu ostatnich lat obserwujemy zmianę podejścia do projektowania procesorów spowodowaną ograniczeniami fizycznymi technologii CMOS. Problemy ze zwiększaniem częstotliwości taktowania spowodowały, że zaczęły się pojawiać układy z bardzo duża liczbą rdzeni obliczeniowych.

Jako cel pracy doktorskiej postanowiono opracowanie metod umożliwiających efektywną realizację algorytmów oddziałujących cząstek, dedykowanych dla wybranych, nietypowych architektur komputerowych.

Pierwszą grupą architektur były procesory graficzne (GPU), które w ciągu ostatnich lat rozwinęły się od prostych układów wykonujących jedynie predefiniowany zestaw operacji arytmetycznych do układów charakteryzujących się wielopotokowością oraz wysoką programowalnością. Kolejną chronologicznie architekturą był procesor Cell opracowany na potrzeby konsoli do gier Playstation 3, a występujący obecnie również w superkomputerach z listy TOP500. Układ składa się z jednego rdzenia ogólnego przeznaczenia oraz ośmiu pomocniczych, przeznaczonych do przetwarzania dużej ilości danych. Ostatnią omawianą architekturą jest CUDA, rozwinięcie i uogólnienie koncepcji obliczeń na GPU do zastosowań, które nie są związane z grafiką komputerową. Do tej kategorii należy procesor Tesla oraz ostatnie generacje kart graficznych.

W pracy wysunięto tezę, że opracowanie dedykowanych algorytmów dla architektur masywnie równoległych pozwoli osiągnąć większą wydajność, niż na architekturach tradycyjnych. Dla wykazania prawdziwości tezy postanowiono zrealizować następujące cele:

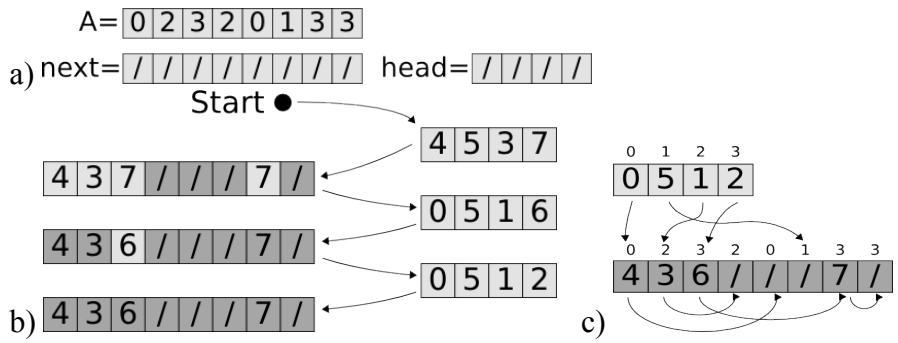
1. Dokonano analizy modelu oddziaływań wykorzystywanych wykorzystywanych w symulacjach metodami cząstek pod kątem ich efektywnej realizacji.
2. Opracowano i zaimplementowano algorytmy symulacji dla wybranych architektur.
3. Przeprowadzono badanie algorytmów pod kątem złożoności obliczeniowej oraz wydajności opracowanych metod.
4. Następnie dokonano symulacji wybranych zjawisk fizycznych za pomocą opisanych metod.

Procesory graficzne

Symulacja metodami cząstek, takimi jak DPD czy SPH, realizowana jest przez wielokrotne powtórzenie kroku symulacyjnego, który polega na wyliczeniu sił oddziaływań między cząstkami, a następnie wyliczenia z równań ruchu nowych położeń cząstek. Dobór metod całkowania nie należał do przedmiotu rozprawy, dlatego zastosowane zostały popularne i sprawdzone rozwiązania.

Ze względu na strumieniowy model przetwarzania główną trudność w implementacji metod cząstek na procesorach graficznych stanowiło wyznaczenie cząstek sąsiednich każdej cząstki, konieczne dla wyliczenia oddziaływań. Autor zastosował podział przestrzeni symulacji (prostopadłościennego pudła obliczeniowego) na sześcienne komórki. Dla danej cząstki należało w takim układzie przejrzeć wszystkie cząstki z 27 komórek otaczających daną cząstkę sprawdzając jednocześnie, czy znajdują się one w tzw. promieniu obcięcia. Stąd długość promienia obcięcia nie może być większa od długości krawędzi celi.

W celu przyporządkowania cząstek do komórek postanowiono zastosować **zmodyfikowany algorytm sortowania kubełkowego** (rys.1.). Celem działania algorytmu było wyznaczenie list odsyłaczowych elementów dla każdej komórki. Uzyskiwane było to przez naprzemienne zapisywanie identyfikatorów elementów do "głowy" listy, a następnie wyznaczanie elementu następnego. Ze względu na konflikty w zapisie, powodowane przypisaniem wielu elementów do tej samej komórki, kroki te musiały być powtarzane wielokrotnie aż do umieszczenia wszystkich elementów w listach. Element raz umieszczony w częściowej liście odsyłaczowej nie był brany pod uwagę w dalszej części przetwarzania.



Rys.1. Przykład działania równoległej wersji algorytmu sortowania kubełkowego: a) dane wejściowe, b) poszczególne kroki przetwarzania, c) wynik sortowania.

Złożoność obliczeniowa proponowanego algorytmu jest rzędu $O(N^2/M)$ gdzie N i M to odpowiednio liczba cząstek i liczba kubełków. Ze względu na równoległe przetwarzanie liczba cykli procesora mogła być mniejsza, niż dla klasycznego algorytmu sortowania kubełkowego.

Dla powyższego algorytmu przeprowadzono dwie **dodatkowe optymalizacje**. Pierwszą z nich było zastosowanie "przesuwanego okna" o stałym rozmiarze. Przy pierwszym przebiegu listy odsyłaczowe tworzone były jedynie w obrębie okien, a scalane w jedną były dopiero w ostatecznym kroku algorytmu.

Druga optymalizacja wynikała z obserwacji, że w ciągu kroku czasowego symulacji tylko niewielka część cząstek zmieniała zajmowaną komórkę. W związku z powyższym dodano początkowy krok, którego zadaniem było usunięcie z gotowych list odsyłaczowych elementów, które powinny znaleźć się w innych komórkach. Następnie stosowany był wyżej opisany algorytm sortowania kubełkowego, z tą różnicą, że zaczynał od już częściowo uporządkowanych list.

Zaproponowany **algorytm symulacji został zaimplementowany** na karcie graficznej GeForce 6800 Ultra. Dane cząstek, położenie i pęd, przechowywane były w postaci 32-bitowych,

zmiennoprzecinkowych tekstur, natomiast indeksy list odsyłaczowych kodowane były jako 8-bitowe, całkowite składowe RGBA.

Implementacja algorytmu na GPU wymagała rozwiązania szeregu problemów technicznych. Ze względu na model przetwarzania potokowego nie było możliwe zrealizowanie zapisu do dowolnej pozycji w pamięci. W tym celu autor zastosował mechanizm renderowania punktów. Dane cząstek były kopiowane z pamięci tekstury do pamięci wierzchołków, skąd następowało renderowanie punktów z przetwarzaniem pozycji cząstki na identyfikator komórki przy użyciu jednostki *vertex shader*.

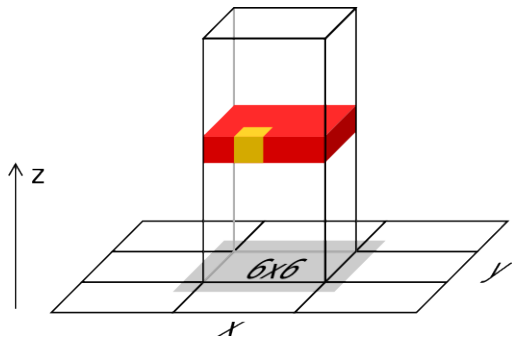
Drugą przeszkodą była realizacja warunku stopu. Do tego celu zastosowano mechanizm testowania okluzji, informującej o liczbie elementów, które pozytywnie przeszły wszystkie testy w potoku renderowania. W przypadku algorytmu sortowania kubełkowego badane było, ile cząstek dodanych zostało do list odsyłaczowych. Algorytm kończył działanie, gdy liczba ta wynosiła zero.

Dla metody DPD konieczne było również zaimplementowanie na GPU generatora liczb pseudolosowych, działającego jedynie w oparciu o liczby zmiennoprzecinkowe.

Otrzymane **wyniki eksperymentalne** porównano z implementacją na tradycyjnym procesorze oraz podobnym algorytmem sortowania działającym na GPU, pokazując lepszą wydajność algorytmu autorskiego.

Procesor Cell

Układ **Cell jest jednostką heterogeniczną**, składającą się z jednego rdzenia ogólnego przeznaczenie PPU oraz ośmiu niezależnych rdzeni przetwarzania wektorowego SPU (sześciu w przypadku wersji procesora w Playstation 3), każdy wyposażony jedynie w 256kB dostępnej pamięci. Ze względu na nietypową budowę procesora standardowy schemat realizacji symulacji metodami cząstek polegający na: 1) znalezieniu par oddziałujących cząstek, 2) obliczeniu sił oddziaływań, 3) rozwiązania układu równań ruchu musiał zostać zmodyfikowany, aby wykorzystać dostępną moc obliczeniową oraz specyficzną architekturę pamięci. Ze względu na duża liczbę obliczeń zmiennoprzecinkowych oraz lokalność oddziaływań obliczenia te mogły być wykonane na jednostkach SPE. Natomiast pierwszy krok, wymagający przejrzenia wszystkich cząstek, musiał zostać zrealizowany przez rdzeń główny.



Rys.2. Podział przestrzeni symulacji na pionowe „kominy” składające się z poziomych „plastrów” o wysokości komórki. Prezentowany komin składa się z 4x4xN komórek wraz z komórkami otaczającymi.

Z powodów wymienionych powyżej **ogólny układ algorytmu** był następujący: na początku cząstki przyporządkowywane były do odpowiednich komórek na jednostce PPU, następnie dane rozsyłane były do jednostek SPU w celu przeprowadzenia obliczeń. Ostatecznie wyniki były zwracane do pamięci głównej, gdzie następowało scalane.

Cała **przestrzeń symulacji**, składająca się sześciennych komórek, została podzielona na pionowe "kominy", które natomiast składały się z poziomych "plastrów" (rys.2). Podział ten był niezbędny ze względu na niewielką dostępną pamięć jednostek SPU. Po rozdzieleniu cząstek do komórek rdzeń PPU zajmował się rozsyłaniem zadań do jednostek SPU. Podstawową jednostką przesyłaną był "plaster" składający się z NxN komórek "komina", wraz z komórkami otaczającymi. Architektura wewnętrzna procesora Cell pozwalała jednostkom SPU na wykonywanie obliczeń przy równoczesnym, asynchronicznym przesyłaniu danych wejściowych oraz wyjściowych. Ze względu na nieduża pamięć SPU operował równocześnie na trzech "plastrach" oraz buforach wejścia/wyjścia.

Przyjęty **model pamięci** podyktowany był wymogami architektury procesora. W celu zapewnienia efektywnego transferu oraz przetwarzania wektorowego dane były wyrównywane do granicy 128-bitów. Również efektywny transfer danych wymagał wyrównywania początku paczki danych do granicy 128 bajtów.

Program został zaimplementowany w języku C++ z wykorzystaniem instrukcji typu intrinsics, realizujących rozkazy SIMD procesora Cell. Następnie został poddany optymalizacji przez reorganizację operacji arytmetycznych oraz dostępu do pamięci, dając pełniejsze wykorzystanie dostępnych potoków przetwarzania. Przesyłanie danych między pamięcią główną i pamięcią lokalną SPU, inicjowane przez SPU, odbywało się asynchronicznie.

Opisany **algorytm został zaimplementowany i uruchomiony** na konsoli do gier Playstation 3, wyposażonej w procesor Cell, działającej pod kontrolą systemu operacyjnego Linux. Zbadano wydajność programu symulacyjnego dla różnych konfiguracji oraz różnej liczby aktywnych rdzeni. Wyniki porównano z tradycyjnym procesorem, wykazując lepszą wydajność procesora Cell. Dodatkowo zbadano zysk z zastosowanych optymalizacji, jednak całkowite przyspieszenie było bardzo niewielkie rzędu 3% procent.

W ramach pracy przeprowadzono również badanie wydajności przesyłania danych między jednostką główną a jednostkami pomocniczymi dla użytego schematu komunikacji. Uzyskano wynik maksymalny rzędu 13GB/s.

CUDA

Architektura CUDA powstała w wyniku rozwoju procesorów graficznych, pojawiła się w momencie, gdy układy te zaczęły zyskiwać uznanie w zastosowaniach naukowych o charakterze obliczeniowym. CUDA znosi konieczność stosowania interfejsów programowania grafiki oraz liczne ograniczenia z nimi związane, jednak nadal wykazuje duże podobieństwo do klasycznej architektury GPU. W związku z tym ogólny układ algorytmu jest podobny do omówionego w ramach GPU. Tutaj także głównym problemem była implementacja algorytmu znajdowania sąsiadów, realizowanego za pomocą sortowania kubełkowego.

Algorytm sortowania kubełkowego dla architektury CUDA różni się jednak zasadniczo od omawianego powyżej. Nowością wprowadzoną do architektury CUDA były instrukcje zapewniające

atomowość równoległych operacji na pamięci. Za pomocą instrukcji atomicExch możliwe jest zrealizowanie sortowania kubełkowego w kilku instrukcjach. W pierwszym kroku na początek listy wpisywany jest element, należący do kubełka, a następnie ustawiany jest wskaźnik na element następny, którym jest dotychczasowy element z początku. Ze względu na atomowość tych operacji nigdy nie nastąpi konflikt w zapisie. Natomiast rozwiązanie takie jest mało wydajne ze względu na koszt operacji atomowych oraz nie wykorzystuje w pełni dostępnego sprzętu.

Dlatego też **dokonano szeregu optymalizacji** i zmian. Po pierwsze należy zauważyć, że wątki wykonywane są grupami po kilkaset. W związku z tym korzystne jest, aby pracowały na wspólnych danych. CUDA oferuje pamięć współdzieloną dla grup wątków, gdzie kopiowane były dane do przetwarzania. Aby zapewnić spójność danych w pamięci wspólnej dokonywana jest synchronizacja dostępów. Algorytm realizuje następujące kroki:

1. Kopiowanie danych do pamięci wspólnej, gdzie budowana będzie lokalna lista odsyłaczowa.
2. Znalezienie elementów należących do tych samych kubełków w lokalnych danych i tworzenie lokalnych list odsyłaczowych.
3. Integrowanie lokalnych list odsyłaczowych z danymi globalnymi, z wykorzystaniem operacji atomowych.
4. Ustawienie końca listy lokalnej na element będący uprzednio na przedzie.
5. Kopiowanie lokalnej tablicy indeksów do pamięci globalnej.

Problemem z wykorzystaniem listy odsyłaczowej jest rozproszenie danych w pamięci. Dlatego wprowadzony został **krok reorganizacji**, który przekształcał listę odsyłaczową w tablicę. Numer kubełka elementów był niemalejący. Aby tego dokonać należało równolegle wyliczyć rozmiar każdego z kubełków, a następnie wyznaczyć indeksy początkowe dla każdego kubełka. Na zakończenie następowała reorganizacja danych według otrzymanych indeksów.

Opisany algorytm sortowania był podstawą do implementacji metod oddziałujących cząstek. Dla danej cząstki obliczenie oddziaływań wymagało przejrzenia list cząstek z otaczających komórek. Następnie nowe współrzędne położenia wyznaczane były z równań ruchu.

Otrzymane wyniki eksperymentalne porównano z istniejącym algorytmem sortowania pozycyjnego. Algorytm prezentowany przez autora dla dużych danych był szybszy od konkurencyjnego o 20-40%. Następnie porównano czasy symulacji na karcie graficznej GeForce 8800GT z procesorem tradycyjnym.

Wyniki eksperymentalne

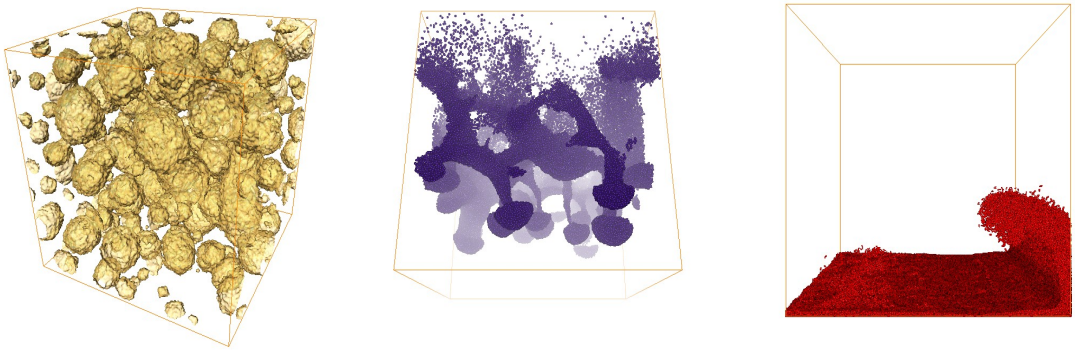
Tab.1. pokazuje porównanie czasów obliczeń dla różnych architektur komputerowych. Celem pracy doktorskiej było opracowanie i implementacja algorytmów oddziałujących cząstek dla pierwszych trzech: GPU, Cell i CUDA. Ze względu na różny czas, w którym pojawiały się te procesory nie jest wskazane porównanie ich między sobą. Widać, że wraz z pojawianiem się nowych generacji sprzętu czas symulacji malał. Dla porównania zostały pokazane czasy symulacji dla dwóch tradycyjnych procesorów. Procesor Itanium został umieszczony w porównaniu ze względu na powszechne użycie w klastrach takich jak Altix. Pokazano wyniki również dla nowoczesnego procesora, jakim był Core 2 Duo. Procesor ten wykonywał program jednowątkowy, dlatego należy przypuszczać, że czas obliczeń wielowątkowych byłby o połowę krótszy. Nie mniej jednak czas uzyskany algorytmami opracowanymi

przez autora jest znacząco krótszy.

Na rys.3. przedstawiono wybrane klatki z symulacji metodami cząstek, które były realizowane w ramach pracy doktorskiej. Jakościowe wyniki nie różniły się między poszczególnymi architekturami, natomiast różne były trajektorie pojedynczych cząstek. Wynikało to z różnej kolejności obliczeń przeprowadzanych przez zaimplementowane algorytmy.

Procesor	Czas obliczeń [s]
GPU (GeForce 6800 Ultra)	1.868
Cell B.E.	1.353
CUDA (GeForce 8800 GT)	0.251
Itanium 1.5 GHz	15.765
Core 2 Duo 2.4 GHz (1 wątek)	3.392

Tab. 1. Czas obliczeń jednego kroku symulacyjnego w sekundach dla różnych architektur komputerowych. Układ obliczeniowy składał się z 2^{20} cząstek.



Rys.3. Przykładowe wyniki symulacji uzyskane w ramach badania algorytmów opisanych w pracy. Od lewej do prawej: separacja faz (DPD), efekt Rayleigha-Taylora (DPD), przerwanie tamy (SPH).

Podsumowanie

W rozprawie przedstawione zostały wyniki prac nad zastosowaniem architektur masywnie równoległych do rozwiązywania problemów symulacji metodą oddziałujących cząstek. Przedstawione propozycje algorytmów i ich implementacje dotyczyły oddziaływań krótkozasięgowych występujących w modelach takich, jak dynamika dyssypatywnych cząstek (DPD) oraz także hydrodynamika wygładzonych cząstek (SPH). Metody tego typu wymagają dużych mocy obliczeniowych. Jednak bezpośrednie przenoszenie istniejących algorytmów na architektury masywnie równoległe nie jest, z wielu powodów, możliwe. Konieczna jest nie tylko zmiana algorytmów ale także koncepcji projektowania oprogramowania i specyficzna implementacja.

W niniejszej pracy udało się zrealizować następujące cele:

- Udowodniono możliwość przeprowadzania efektywnej symulacji metodami cząstek na architekturach masywnie wielopotokowych. Dotyczy to także jednostek obliczeniowych, takich jak karty graficzne, których oryginalne przeznaczenie jest zupełnie inne.
- Zbadano funkcjonalność oraz model programowania pod kątem symulacji metodami cząstek dla następujących architektur komputerowych: kart graficznych korzystających z graficznego interfejsu programowania (np. OpenGL), heterogenicznego procesora Cell Broadband Engine, jednostek o architekturze CUDA.
- Zaproponowano implementację symulacji metodami cząstek dla każdej z wymienionych architektur.
- Opracowano algorytmy sortowania kubełkowego oraz korzystające z nich algorytmy znajdowania sąsiadów dla architektur GPU oraz CUDA.
- Zbadano wydajność oraz określono złożoność obliczeniową algorytmów zaimplementowanych dla testowanych architektur. Wyniki zostały porównane z wynikami otrzymanymi dla tradycyjnych architektur procesorowych. Świadczą one o przydatności architektur masywnie równoległych dla symulacji metodami cząstek. Co więcej, zaproponowane algorytmy uzyskują krótsze czasy wykonania niż ich odpowiedniki na tradycyjnych procesorach.
- Dla architektury CUDA porównane zostały różne sposoby znajdowania i przeglądania sąsiadów. Wskazano metodę optymalną.
- Dla architektury Cell zaproponowano efektywny mechanizm podziału problemu obliczeniowego na mniejsze zadania umożliwiający wydajną transmisję danych pomiędzy jednostkami składowymi procesora. Ponadto przebadano przepustowość przesyłania danych z pamięci systemowej do pamięci lokalnych.
- Dla kart graficznych zaproponowane zostało wykorzystanie generatora liczb pseudolosowych, operującego jedynie na liczbach zmiennoprzecinkowych.

Badania przeprowadzane w ramach niniejszej pracy były ograniczone pod różnymi względami. Pozostawiają one jednak miejsce na dalszą pracę, w ramach której można wymienić:

- Implementację nowoczesnych modeli oddziałujących cząstek uwzględniających stan termodynamiczny układu takich, jak GENERIC DPD oraz SDPD.
- Wzbogacenie implementowanych modeli oddziaływań cząstek o oddziaływania dalekiego zasięgu.
- Zaproponowanie wspólnego środowiska dla realizacji różnorodnych metod cząstek, które potrafiłoby efektywnie wykorzystać różnego typu sprzęt komputerowy.
- Opracowanie algorytmów dla wykorzystania większej liczby węzłów obliczeniowych. Klaster złożony z komputerów o jednostkach masywnie wielopotokowych byłby w stanie rozwiązywać problemy o niespotykanych dotychczas rozmiarach.
- Przetestowanie architektur komputerowych następnych generacji, takich jak Larrabee firmy Intel.