Tạp chí Khoa học ĐHQGHN, Khoa học Tự nhiên và Công nghệ 28 (2012) 147-157

Xây dựng cây bát phân dựa trên đường cong lấp đầy không gian với bộ xử lý đồ họa GPU

Nguyễn Hải Châu*

Trường Đại học Công nghệ, ĐHQGHN, 144 Xuân Thủy, Hà Nội, Việt Nam

Nhận ngày 5 tháng 12 năm 2011

Tóm tắt. Cây bát phân (octree) là một cấu trúc dữ liệu có nhiều ứng dụng trong các lĩnh vực như đồ họa máy tính, mô phỏng và mô hình hóa. Trong mô phỏng động lực phân tử và mô phỏng *N*-body, cây bát phân được sử dụng nhiều với các thuật toán phân cấp như tree, khai triển đa cực nhanh FMM (fast multipole method) để tính lực tương tác xa. Có nhiều phương pháp và cấu trúc dữ liệu có thể sử dụng để xây dựng cây bát phân. Trong bài báo này, chúng tôi sử dụng đường cong lấp đầy không gian SFC (space-filling curve) để xây dựng cây bát phân và song song hóa thuật toán xây dựng cây trên bộ xử lý đồ họa GPU. Kết quả thực nghiệm trên máy tính Acer 5745G với bộ xử lý Intel Core i3 2.13 GHz, 4GB RAM được trang bị bộ xử lý đồ họa GeForce 310M và máy tính để bàn với CPU Dual-Core AMD Opteron 2216 HE 1.0 GHz, 2GB RAM, được trang bị bộ xử lý đồ họa nVidia GeForce 8800 GT cho thấy, thuật toán song song do chúng tôi cài đặt trên GPU có tốc độ thực hiện nhanh hơn thuật toán tuần tự trên CPU từ 2.1 đến 7 lần đối với các hệ có từ 2^{20} đến 10.2^{20} phân tử. Đồng thời, mức độ tăng tốc của thuật toán song song xây dựng cây bát phân phụ thuộc vào hai yếu tố chính, đó là tốc độ truyền dữ liệu giữa máy tính với GPU và tốc độ thực hiện thuật toán sắp xếp các phân tử theo khóa Morton trên GPU.
Từ khóa: cây bát phân (octree), FMM, treecode, mô phỏng động lực phân tử, mô phỏng *N*-body, GPU.

1. Mở đầu

Mô phỏng động lực phân tử [1] là một trong các phương pháp phổ dụng để nghiên cứu các hệ vật lý hóa học với sự trợ giúp về tính toán của máy tính điện tử. Trong mô phỏng động lực phân tử, việc tính toán lực tương tác xa rất tốn thời gian và thường chiếm trên 90% thời gian thực hiện của các mô phỏng. Chính vì vậy, đã có nhiều giải pháp nhằm tăng tốc độ tính toán lực tương tác xa. Các phương pháp này thường chia thành 3 loại chính: Áp dụng các thuật toán có độ phức tạp $O(N)$ hoặc $O(N\log N)$ [2-7], với N là số lượng phân tử của hệ; sử dụng phần cứng tốc độ cao như máy tính song song, bộ xử lý đồ họa GPU hoặc phần cứng được thiết kế chuyên dụng để tính lực tương tác như GRAPE [8, 9]; và kết hợp hai giải pháp nói trên.

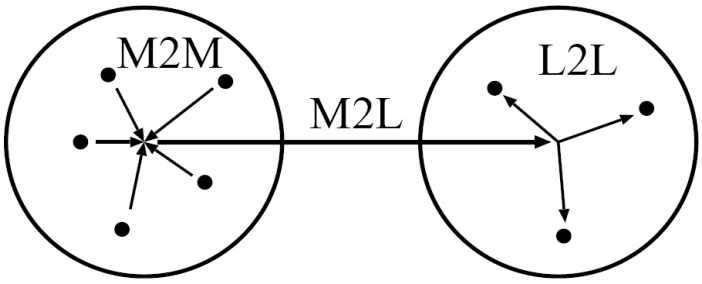
Thuật toán đơn giản và chính xác nhất để tính lực tương tác xa, đó là tính lực tương tác theo từng cặp đôi. Thuật toán có độ phức tạp tính toán $O(N^2)$, thường chỉ được sử dụng cho

* ĐT: 84-4-37547813.
E-mail: chaunh@vnu.edu.vn

các hệ có số lượng phân tử N nhỏ hơn 10^5. Với các hệ có số lượng lớn hơn, thuật toán này không hiệu quả do thời gian tính toán quá lâu.

Năm 1987, Greengard và Rokhlin đã phát minh ra thuật toán khai triển đa cực nhanh (fast multipole method – FMM) có độ phức tạp tính toán $O(N)$ trong không gian hai chiều [3]. Đến năm 1997, thuật toán FMM trong không gian 3 chiều được công bố đánh dấu bước tiến lớn trong việc tính toán nhanh lực tương tác xa [4]. Từ đó đến nay, thuật toán FMM được sử dụng rộng rãi và có nhiều biến thể khác nhau như Anderson [2], Makino [10], Ying, Biros và Zorin [11].

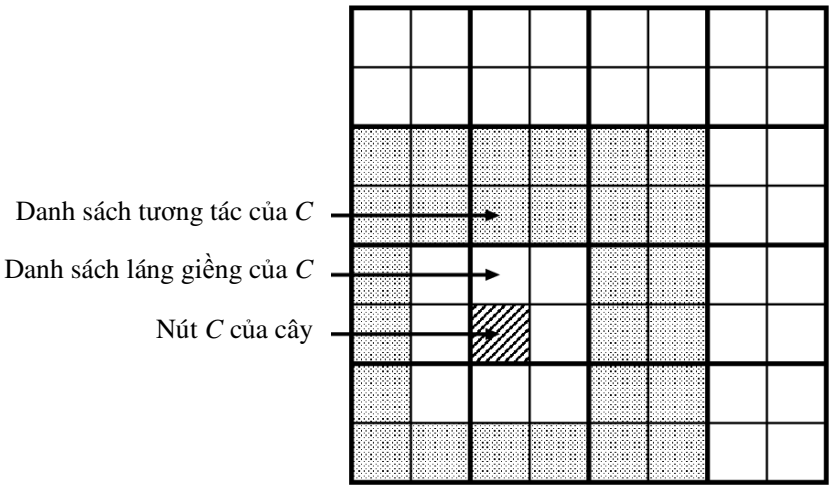
Khi thực hiện các mô phỏng động lực phân tử với hầu hết các thuật toán trong đó có FMM, các phân tử thường được phân bố trong một hình lập phương với cạnh có độ dài 1 trong không gian ba chiều. Để tính toán lực tương tác xa giữa các phân tử, thuật toán FMM tính toán tương tác giữa các nhóm phân tử, sau đó sử dụng khai triển Taylor và khai triển đa cực để tính xấp xỉ lực tác động vào các phân tử với độ chính xác được xác định trước. Hình 1 minh họa ý tưởng của thuật toán FMM.



Hình 1. Minh họa ý tưởng của thuật toán FMM. **M2M**: Biến đổi khai triển đa cực – đa cực, **M2L**: Biến đổi khai triển đa cực – Tay lor, **L2L**: Biến đổi khai triển Taylor – Taylor.

Để thực hiện được các biến đổi M2M, M2L và L2L, FMM sử dụng cấu trúc dữ liệu cây bát phân. Cây bát phân trong thuật toán FMM được xây dựng căn cứ vào vị trí của các phân tử trong hình lập phương như đã nói trên. Nút gốc của cây bát phân chính là hình lập phương chứa toàn bộ các phân tử. Tám nút con của mỗi nút trong cây bát phân được tạo bằng cách chia đều nút cha thành 8 hình lập phương bằng nhau. Một phân tử có vị trí nằm trong một hình lập phương sẽ được gán cho nút tương ứng với hình lập phương đó. Quá trình tạo cây bát phân dừng khi cây đạt đến một độ sâu l định trước.

Khi thực hiện thuật toán FMM, các biến đổi M2M được thực hiện trong quá trình duyệt cây bát phân theo chiều rộng và từ lá đến gốc. Các biến đổi M2M được thực hiện cho toàn bộ các nút ở mức l, sau đó cho toàn bộ các nút ở mức l-1... cho đến mức 0. Các biến đổi M2L và L2L được thực hiện trong quá trình duyệt cây bát phân theo chiều rộng và từ gốc đến lá. Bởi vậy tốc độ duyệt cây bát phân và việc xác định các phân tử thuộc những nút nào trong cây ở mỗi mức sâu có ảnh hưởng lớn đến hiệu năng của thuật toán FMM.



Hình 2. Danh sách láng giềng và tương tác.

Trong quá trình thực hiện FMM, tại các pha M2L và pha tính lực của thuật toán chúng ta bắt buộc phải tính toán danh sách láng giềng và danh sách tương tác của mỗi nút trong cây bát phân tại mỗi mức độ sâu. Một nút được xem là láng giềng của một nút ở cùng độ sâu nếu hai nút này có tiếp xúc về mặt hình học – tức là có thể có tiếp xúc về mặt, cạnh hoặc đỉnh. Một nút được xem là nằm trong danh sách tương tác của một nút ở cùng độ sâu nếu hai nút cha của chúng là láng giềng. Hình 2 minh họa danh sách láng giềng và tương tác trong không gian hai chiều (cây tứ phân). Trong không gian hai chiều, một nút có nhiều nhất 8 láng giềng và trong không gian 3 chiều, mỗi nút có nhiều nhất 26 láng giềng.

Với các yêu cầu tính toán khi duyệt cây bát phân như mô tả ở trên, cấu trúc dữ liệu đệ qui truyền thống để biểu diễn cây bát phân là không thích hợp và làm giảm hiệu năng của quá trình duyệt cây để tính toán các biến đổi M2M, M2L và L2L do phải tìm kiếm để xác định các danh sách láng giềng và tương tác [3,4].

Trong bài báo này, chúng tôi sử dụng bộ xử lý đồ họa GPU để cài đặt thuật toán xây dựng cây bát phân trong thuật toán FMM với các mục tiêu chính là tăng tốc độ xây dựng cây bát phân, tăng tốc độ thực hiện việc duyệt cây theo hai chiều từ lá đến gốc và gốc đến là, tăng tốc độ các phép tìm kiếm: Tìm nút cha và nút con của một nút trong cây, tìm danh sách hàng xóm và danh sách tương tác, tìm nút chứa một phân tử bất kỳ. Các phép tìm kiếm này được thực hiện nhiều lần trong thuật toán FMM. Các phần còn lại của bài báo được tổ chức như sau: Phần hai mô tả tóm tắt về GPU, phần 3 trình bày về cây bát phân các cấu trúc dữ liệu liên quan và phương pháp xây dựng cây bát phân trên GPU của chúng tôi. Phần 4 là thảo luận, phân tích hiệu năng và cuối cùng là phần kết luận.

2. Bộ xử lý đồ họa GPU

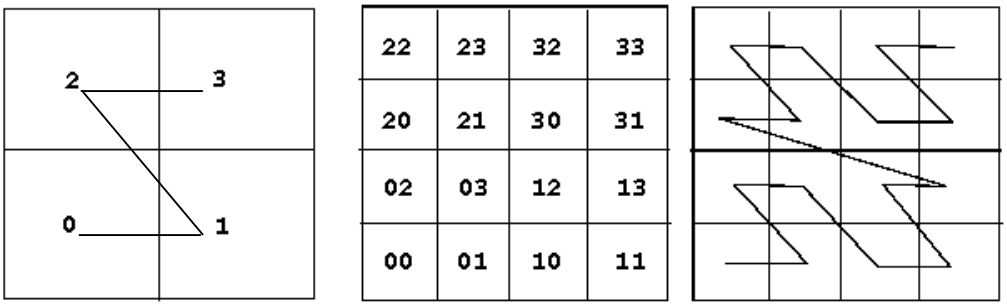
Bộ xử lý đồ họa (Graphics Processing Unit – GPU) là bộ xử lý chuyên dụng được thiết kế dành riêng để thực hiện các tác vụ đồ họa trong máy tính nhằm tăng tốc các ứng dụng đồ họa

nói chung. Từ năm 2007, nVidia [12] – một trong các hãng đi đầu về sản xuất GPU – đã giới thiệu các bộ xử lý đồ họa GPU có thể sử dụng để thực hiện các bài toán thông dụng (general-purpose computing) bên cạnh việc xử lý các tác vụ đồ họa truyền thống. Với kiến trúc đa nhân, cho phép lập trình theo mô hình song song dữ liệu (data parallel), GPU đã dần trở thành một công cụ tính toán hiệu năng cao với giá rẻ và được sử dụng nhiều trong lĩnh vực tính toán hiệu năng cao.

Để có thể sử dụng GPU cho tính toán hiệu năng cao, người lập trình chỉ cần có một bộ GPU có hỗ trợ tính toán thông dụng (general-purpose computing) [13], một máy tính có cài đặt phần mềm điều khiển GPU (gọi tắt là máy chủ) và bộ phát triển phần mềm với GPU. GPU nhận lệnh, dữ liệu từ máy chủ; tính toán và trả lại kết quả cho máy chủ. Tốc độ tính toán của GPU thường nhanh hơn máy chủ cá nhân thông thường từ hàng chục đến hàng trăm lần [14]. Hiện tại có hai loại GPU thường được sử dụng cho tính toán thông dụng, đó là các sản phẩm của các hãng nVidia và AMD [15]. Trong bài báo này chúng tôi sử dụng GPU do hãng nVidia sản xuất để cài đặt thuật toán xây dựng cây bát phân song song.

3. Cây bát phân và đường cong lấp đầy không gian

Có nhiều phương pháp cài đặt cây bát phân sử dụng các cấu trúc dữ liệu khác nhằm làm tăng tốc độ của quá trình tạo, duyệt cây và tính toán các danh sách láng giềng, tương tác của mỗi nút. Một trong các cấu trúc dữ liệu thường được sử dụng là đường cong lấp đầy không gian (space-filling curve - SFC) [16], nhằm mô tả cây bát phân dưới dạng cấu trúc dữ liệu không đệ qui – chính là các mảng một chiều. Có nhiều loại SFC như Morton, Hilbert-Peano, Siepinsky. Để dễ hình dung, chúng tôi minh họa đường cong Morton trong không gian hai chiều và mối quan hệ của nó với cây tứ phân trên Hình 3 (tương tự như cây bát phân trong không gian ba chiều).



Hình 3. Cây tứ phân và đường cong lấp đầy không gian Morton ở mức 1, 2.
Các số biểu diễn thứ tự của các nút trong SFC tương ứng.

Chúng tôi đã sử dụng SFC để cài đặt cây bát phân [17-19], tuy nhiên quá trình tính toán các danh sách láng giềng và tương tác chưa đạt được hiệu quả cao; đồng thời quá trình xây dựng cây và duyệt cây vẫn thực hiện hoàn toàn trên máy chủ do đó hiệu năng xây dựng cây bị hạn chế. Trong bài báo này, chúng tôi khắc phục các hạn chế nói trên, đồng thời cài đặt thuật toán xây dựng cây bát phân song song trên GPU.

3.1. Đường cong lấp đầy không gian Morton và cây bát phân

Để biểu diễn cây bát phân bằng đường cong lấp đầy không gian Morton, chúng ta cần tính toán khóa Morton cho mỗi nút của cây, đồng thời cần biến đổi được vị trí của mỗi phân tử trong hệ phân tử cần mô phỏng về dạng một khóa Morton.

Việc tính toán khóa Morton cho mỗi nút của cây tương đối đơn giản. Xuất phát từ nút gốc với khóa Morton 0, chúng ta xây dựng khóa Morton cho các nút ở mức 1, 2 như mô tả trên Hình 3. Đối với việc tính toán khóa Morton cho mỗi vị trí của các phân tử trong hệ cần mô phỏng, chúng ta ánh xạ vị trí của (x, y, z) phân tử vào khóa Morton tương ứng theo cách sau:

- Xác định trước độ dài k của khóa Morton và k thường được nhận giá trị từ 16 đến 21 bit. Người ta thường xác định k căn cứ theo độ dài của từ máy (32 hay 64 bit) và số lượng phân tử trong hệ cần mô phỏng. Giá trị k càng lớn thì khả năng trùng lặp khóa của hai phân tử khác nhau càng nhỏ, nhưng thời gian xây dựng cây bát phân và bộ nhớ dành cho cây bát phân càng lớn.

- Tính khóa Morton m_x, m_y, m_z tương ứng cho mỗi tọa độ x, y, z.

- Ghép các bit ở cùng vị trí của ba khóa Morton riêng lẻ m_x, m_y, m_z để tạo ra khóa Morton chung cho vị trí phân tử (x, y, z). Do hiện nay độ dài từ máy cao nhất là 64 bit nên để tạo được khóa Morton cho (x, y, z), mỗi khóa Morton riêng lẻ chỉ có thể có nhiều nhất 21 bit.

Việc tính toán khóa Morton cho các nút của cây và cho vị trí các phân tử (hàm **particle_mortonkey**) được cài đặt trên ngôn ngữ C như sau:

```
unsigned long mortonkey(unsigned long x, int keylen)
{
	MORTONKEY	tmp = 0;
	int	i, j;

	j = 0;
	for (i=0;i<keylen;i++) { // for each low bit i-th from 0..keylen-1
		if ((x>>i) & 1){ // if the bit is 1
			tmp |= ((unsigned long)1)<<j; // then set bit i*3 to 1
		}
		j += 3;
	}
	return(tmp);
}
unsigned long
particle_mortonkey(double *pos, double rscale, unsigned long offset, int keylen)
{
	unsigned long	x[3];
	int	i;
	for (i=0;i<3;i++){
		x[i] = (unsigned long)(pos[i]*rscale+offset);
	}
```

```
return((mortonkey(x[0],keylen)<<2) | (mortonkey(x[1],keylen)<<1) | (mortonkey(x[2
],keylen)));
}
```

Khi đã tính toán được khóa Morton cho các nút của cây và các phân tử, chúng ta có thể dễ dàng xác định các đối tượng sau đây nếu biết trước khóa Morton *m* của một nút ở mức *i*:

- Nút cha của nút này ở mức *i*-1, có khóa Morton là *m* dịch phải 3 bit. Độ phức tạp của tác vụ tính toán này là O(1).

- Danh sách các nút con của *m*: Gọi *n* = (*m* dịch trái 3 bit). Khi đó *m* có 8 nút con ở mức *i*+1 có khóa Morton lần lượt là *n*, *n*+1,... *n*+7. Độ phức tạp của tác vụ tính toán là O(1).

- Danh sách các láng giềng của *m*: Độ phức tạp của phép tính toán này là O(d^3), với *d* là số chiều của không gian đang xét của các phân tử, *d* thường có giá trị 3; từ đó dễ dàng xác định được danh sách tương tác của *m* cùng với độ phức tạp tính toán O(d^3).

Ngoài ra, nếu biết vị trí (*x*, *y*, *z*) của một phân tử, ta có thể nhanh chóng xác định khóa Morton *m* của nút mức *i* chứa nó với các mã lệnh C sau, trong đó *k* là độ dài khóa Morton:

```
k0 = (long)(x*((unsigned long)1<<i));
k1 = (long)(y*((unsigned long)1<<i));
k2 = (long)(z*((unsigned long)1<<i));
m = (mortonkey(k0,k)<<2) | (mortonkey(k1,k)<<1) | (mortonkey(k2,k));
```

3.2. Thuật toán xây dựng cây bát phân song song trên GPU

Với cách tính toán khóa Morton như mô tả ở trên và cách tổ chức cây bát phân theo kiểu con trỏ đến các mảng ở từng mức sâu, thuật toán tuần tự xây dựng cây bát phân gồm có các bước chính như sau:

(1) Xác định độ sâu *l* của cây, sau đó cấp phát bộ nhớ cho các nút của cây: Mỗi mức sâu *i* tương ứng với một mảng có 8^i phần tử.

(2) Tính toán khóa Morton cho tất cả các phân tử trong hệ cần mô phỏng.

(3) Sắp xếp các phân tử trong hệ theo thứ tự tăng dần của khóa Morton bằng thuật toán quicksort.

(4) Căn cứ vào thứ tự của các phân tử theo khóa Morton, chia các phân tử vào các nút của cây. Do các phân tử đã được sắp xếp theo thứ tự của khóa Morton, mỗi nút của cây có cấu trúc dữ liệu ghi nhớ vị trí của hai phân tử đầu và cuối.

Trong các bước nói trên, bước 2, 3 và 4 đều có thể song song hóa được theo mô hình song song hóa dữ liệu (data parallel). Mô hình này rất phù hợp với kiến trúc phần cứng của các đơn vị xử lý đồ họa GPU. Do đó, để song song hóa quá trình xây dựng cây với GPU, chúng tôi đã xây dựng các hàm kernel (thực hiện trên GPU) sau đây:

- Hàm kernel tính khóa Morton gpu_morton_body của một phân tử dựa trên vị trí (*x*, *y*, *z*) trong không gian ba chiều của phân tử đó.

- Hàm kernel xây dựng cây bát phân gpu_octree_calc.

Đồng thời để song song hóa thuật toán sắp xếp các phân tử ở bước 2, chúng tôi sử dụng thư viện thrust có trong bộ phát triển phần mềm tính toán trên CUDA trên GPU phiên bản 4.0 của nVidia. Thuật toán xây dựng cây bát phân trên GPU sẽ được mô tả như sau:

(1) Xác định độ sâu l của cây, sau đó cấp phát bộ nhớ cho các nút của cây: Mỗi mức sâu i tương ứng với một mảng có 8^i phần tử.

(2) Tính toán khóa Morton cho tất cả các phân tử trong hệ cần mô phỏng bằng cách thực hiện hàm kernel gpu_mortonkey_body.

(3) Sắp xếp các phân tử căn cứ vào thứ tự của khóa Morton bằng cách sử dụng hàm thư viện thrust::sort_by_key kèm theo bộ phát triển phần mềm với GPU của nVidia.

(4) Chia các phân tử sau khi đã được sắp xếp vào các nút của cây căn cứ vào vị trí của các phân tử đó.

3.3. Môi trường và kết quả thực nghiệm

Chúng tôi đã cài đặt thuật toán xây dựng cây bát phân song song bằng ngôn ngữ C và phần mở rộng của C trên GPU; đồng thời thử nghiệm hiệu năng với hai hệ thống máy tính được trang bị GPU. Hệ thống thứ nhất (gọi tắt là hệ thống 1) là máy xách tay Acer Aspire 5745G với 4GB bộ nhớ trong, CPU Intel Core i3 330M 2 nhân 2.13 GHz, được trang bị bộ xử lý đồ họa nVidia GeForce 330M có 512 MB VRAM. Tốc độ lý thuyết cực đại của nVidia GeForce 330M là 73 GFlop/s. Hệ thống thứ hai (gọi tắt là hệ thống 2) là máy tính để bàn tự lắp đặt với CPU Dual-Core AMD Opteron 2216 HE tốc độ 1.0 GHz, 2GB RAM, được trang bị bộ xử lý đồ họa nVidia GeForce 8800 GT có tốc độ xử lý cực đại 504 GFlop/s. Trong cả hai môi trường thực nghiệm, chúng tôi sử dụng hệ điều hành Linux Fedora phiên bản 14 trở lên, chương trình dịch gcc phiên bản 4.6.1 và bộ phát triển nVidia SDK phiên bản 4.0 cùng với chương trình dịch nvcc phiên bản 4.0 V0.2.1221 [20].

Chúng tôi đã thử nghiệm hiệu năng của thuật toán xây dựng cây bát phân song song trên GPU với vị trí được sinh ngẫu nhiên trong hình lập phương cạnh 1, có tâm tại gốc tọa độ. Số lượng phân tử N được thay đổi từ $1048576=2^{20}$ đến $10485760=10.2^{20}$. Số lượng phân tử từ 2^{20} đến 10.2^{20} là khá lớn cho mô phỏng động lực phân tử hoặc N-body trên một máy tính cá nhân đơn lẻ [11,19]. Trong tất cả các lần thực nghiệm, chúng tôi sử dụng khóa Morton có độ dài 16 bit. Độ sâu l của cây được tính theo công thức $l=\log_8 N$. Trong trường hợp N=10485760, độ sâu của cây là 8.

Dung lượng bộ nhớ của GPU tối thiểu cần sử dụng để thực hiện thuật toán xây dựng cây bát phân bao gồm: Bộ nhớ dành cho vị trí của các phân tử, bộ nhớ dành cho cây bát phân (mỗi nút của cây bát phân có cỡ 32 bytes), bộ nhớ dành cho mảng các khóa Morton của tất cả các phân tử và bộ nhớ dành cho mảng chỉ số khóa – dùng để sắp xếp lại vị trí của các phân tử theo khóa Morton. Trong các thực nghiệm của chúng tôi, do các GPU ở hệ thống 1 và 2 chỉ có thể tính toán với số dấu phảy động độ chính xác đơn (single precision floating point) nên dung lượng bộ nhớ GPU tối thiểu cần có được tính theo công thức:

$$M = N*\text{sizeof(float)} + 32.(8^{l+1}-1) + N*\text{sizeof(unsigned long)}*2$$

trong đó M là dung lượng bộ nhớ tối thiểu, N là số lượng phân tử, l là độ sâu của cây bát phân. M cũng chính là dung lượng dữ liệu tối thiểu cần trao đổi giữa máy chủ và GPU. Trong trường hợp N = 10485760, dung lượng bộ nhớ GPU tối thiểu cần có là $M \approx 449.142$ MB.

Kết quả thực nghiệm được mô tả trong các bảng 1, 2 và 3. Ta có thể thấy rằng thuật toán xây dựng cây bát phân được tăng tốc từ 2.1 đến 3.3 lần trên hệ thống 1 và từ 4.7 đến 7 lần trên hệ thống 2. Trong bảng 3, chúng tôi trình bày kết quả thực nghiệm về việc phân tích thời gian thực hiện của các bước chính khi xây dựng cây bát phân với hai hệ thống 1 và 2 trong trường

hợp số phân tử N=10485760. Kết quả trong bảng 3 được phân tích ở phần 0 của bài báo.

Bảng 1. Thời gian thực hiện thuật toán xây dựng cây bát phân trên hệ thống 1

Số lượng phân tử	Thời gian thực hiện trên máy chủ (giây)	Thời gian thực hiện trên máy chủ có GPU (giây)	Tỷ số tăng tốc GPU / máy chủ (lần)
1048576	0.81	0.38	2.1
2097152	1.72	0.76	2.3
3145728	2.57	1.02	2.5
4194304	3.41	1.24	2.8
5242880	4.28	1.45	3.0
6291456	5.10	1.67	3.1
7340032	5.94	1.88	3.2
8388608	6.79	2.10	3.2
9437184	7.63	2.35	3.2
10485760	8.47	2.57	3.3

Bảng 2. Thời gian thực hiện thuật toán xây dựng cây bát phân trên hệ thống 2

Số lượng phân tử	Thời gian thực hiện trên máy chủ (giây)	Thời gian thực hiện trên máy chủ có GPU (giây)	Tỷ số tăng tốc GPU / máy chủ (lần)
1048576	1.83	0.28	6.5
2097152	3.98	0.85	4.7
3145728	5.90	1.15	5.1
4194304	7.68	1.35	5.7
5242880	9.80	1.40	7.0
6291456	11.57	2.00	5.8
7340032	13.56	2.27	6.0
8388608	15.61	2.57	6.1
9437184	17.70	2.81	6.3
10485760	19.61	3.01	6.5

4. Thảo luận và phân tích hiệu năng

Căn cứ vào bảng 3, chúng tôi đã phân tích hiệu năng cho các thực nghiệm trên và rút ra một số nhận xét như sau.

Trên hệ thống 1, thời gian trao đổi dữ liệu giữa máy chủ và GPU chiếm tỷ lệ 8.9% trong toàn bộ thời gian thực hiện thuật toán xây dựng cây bát phân song song với GPU. Tuy nhiên trên hệ thống 2 thời gian trao đổi dữ liệu chiếm tới 30.8% làm cho hiệu năng toàn bộ của thuật toán giảm đi, mặc dù mức độ tăng tốc của bước tính toán khóa Morton và sắp xếp các phân tử ở hệ thống 2 tương ứng là 74.3 và 29 lần, cao hơn mức độ tăng tốc trên hệ thống 1 rất nhiều (tương ứng là 5.2 và 1.2 lần). Điều đó cho thấy mặc dù tốc độ tính toán cực đại của GPU trên hệ thống 2 cao hơn hệ thống 1 đến 6.9 lần, nhưng do tốc độ trao đổi dữ liệu của hệ thống 2 chậm hơn, dẫn tới hiệu năng chung của toàn bộ thuật toán thực hiện trên hệ thống 1 cao hơn hiệu năng của thuật toán thực hiện trên hệ thống 2. Mặc dù GPU của hệ thống 2 có tốc độ truyền dữ liệu cực đại cao hơn GPU trên hệ thống 1 (theo đặc tả kỹ thuật của nVidia) nhưng do máy chủ của hệ thống 2 có tốc độ bus thấp hơn hệ thống 1 nên tốc độ trao đổi dữ liệu chung của hệ thống 2 thấp hơn hệ thống 1.

Trên hệ thống 2, thời gian sắp xếp các phân tử theo khóa Morton chỉ chiếm 5.9% thời gian thực hiện thuật toán. Trong khi đó trên hệ thống 1, thời gian sắp xếp các phân tử chiếm tới 53.1%. Điều đó cho thấy thời gian thực hiện thuật toán sắp xếp có ảnh hưởng quan trọng đến hiệu năng chung của thuật toán.

Tỷ số tăng tốc của thuật toán song song với GPU có xu hướng tăng lên khi số lượng phân tử N tăng lên. Đây là ưu điểm của thuật toán song song xây dựng cây bát phân với GPU do chúng tôi cài đặt. Sử dụng thuật toán song song hóa cây bát phân có lợi khi số lượng phân tử N lớn.

Bảng 3. Phân tích thời gian (đơn vị tính: giây và tỷ lệ phần trăm) thực hiện các bước trong thuật toán song song xây dựng cây bát phân. Số lượng phân tử N = 10485760.

Các bước thực hiện thuật toán xây dựng cây bát phân sử dụng SFC	Hệ thống 1				Hệ thống 2			
	Thời gian thực hiện trên máy chủ		Thời gian thực hiện trên máy chủ có GPU		Thời gian thực hiện trên máy chủ		Thời gian thực hiện trên máy chủ có GPU	
	Giây	Tỷ lệ	Giây	Tỷ lệ	Giây	Tỷ lệ	Giây	Tỷ lệ
Cấp phát và giải phóng bộ nhớ	0.12	1.4%	0.13	5.0%	0.55	2.8%	0.67	22.2%
Tính toán khóa Morton của tất cả các phân tử	3.45	40.7%	0.66	25.7%	7.49	38.2%	0.10	3.3%
Sắp xếp các phân tử theo khóa Morton đã tính toán	2.14	25.3%	1.37	53.1%	5.17	26.4%	0.18	5.9%
Gán các phân tử cho các nút của cây bát phân	2.76	32.6%	0.19	7.3%	6.40	32.6%	1.14	37.7%
Truyền dữ liệu giữa GPU và máy chủ			0.23	8.9%			0.93	30.8%
Tổng thời gian tạo cây bát phân	**8.47**	100%	**2.57**	100%	**19.61**	100%	**3.01**	100%

5. Kết luận

Chúng tôi đã cài đặt thành công thuật toán xây dựng cây bát phân trên bộ xử lý đồ họa GPU. Với số lượng phân tử từ 2^{20} đến 10.2^{20}, độ sâu của cây ở mức 8 và sử dụng khóa Morton có độ dài 16 bit, xây dựng cây bát phân với bộ xử lý đồ họa GPU đạt tốc độ cao hơn xây dựng cây bát phân trên máy chủ từ 2.1 đến 7 lần. Mức độ tăng tốc của thuật toán phụ thuộc vào hai yếu tố chính, đó là tốc độ trao đổi dữ liệu giữa máy chủ với GPU và tốc độ thực hiện thuật toán sắp xếp các phân tử theo khóa Morton trên GPU. Việc tăng tốc độ cài đặt thuật toán xây dựng cây bát phân và xây dựng các hàm tìm kiếm các danh sách láng giềng và tương tác của một nút trong cây là một trong các yếu tố quan trọng trong việc tăng tốc độ tính lực tương tác giữa các phân tử trong thuật toán khai triển đa cực nhanh FMM.

Lời cảm ơn

Công trình này được tài trợ một phần từ đề tài mang mã số QG.09.27, Đại học Quốc gia Hà Nội. Tác giả chân thành cảm ơn TS Toshiaki Iitaka, phòng thí nghiệm Vật lý Thiên văn Tính toán (Computational Astrophysics Laboratory – CAL [21]) thuộc Viện Vật lý – Hóa học Nhật Bản (RIKEN [22]) đã giúp đỡ, cho phép tác giả được sử dụng hệ thống máy tính của CAL với bộ xử lý đồ họa nVidia GeForce 8800 GT (hệ thống 2) để cài đặt chương trình và tiến hành các thực nghiệm.

Tài liệu tham khảo

[1] J. M. Haile, Molecular Dynamics Simulation: Elementary Methods, *Wiley-Interscience,* 1997.

[2] C. R. Anderson, An implementation of the fast multipole method without multipoles, *SIAM J. Sci. Stat. Comput.* 13 (4) (1992) 923–947.

[3] L.Greengard, V. Rokhlin, A fast algorithm for particle simulations, *Journal of Computational Physics* 73 (1987) 325–348.

[4] L. Greengard, V. Rokhlin, A new version of the fast multipole method for the Laplace equation in three dimensions, *Acta Numerica* 6 (1997) 229–269.

[5] Appel, An efficient program for many-body simulation, SIAM *J. Sci. Stat. Comput.* 6 (1) (1985), 85–103.

[6] J. E. Barnes, A modified tree code: Don't laugh; It runs, *Journal of Computational Physics* 87 (1990) 161–170.

[7] J. E. Barnes, P. Hut, A hierarchical O(N logN) force-calculation algorithm, *Nature* 324 (1986) 446–449.

[8] D. Sugimoto, Y. Chikada, J. Makino, T. Ito, T. Ebisuzaki, M. Umemura, A special-purpose computer for gravitational many-body problems, *Nature* 345 (1990) 33–35.

[9] J. Makino, M. Taiji, *Scientific Simulations with Special-Purpose Computers - The GRAPE Systems* (Chichester: John Wiley and Sons, 1998).

[10] J. Makino, Yet another fast multipole method without multipoles - P^2M^2 multipole method, *Journal of Computational Physics* 151(1999) 910-920.

[11] L. Ying, G. Biros, D. Zorin. A kernel-independent adaptive fast multipole method in two and three dimensions, *Journal of Computational Physics* 196(2004) 591-626.

[12] http://www.nvidia.com

[13] J. Sanders, E. Kandrot, *CUDA by examples – An introduction to general-purpose GPU programming,* Addison-Wesley, 2011.

[14] Wen-Mei W. Hwu ed., *GPU computing gems,* Emerald edition, Morgan-Kaufmann, 2011.

[15] http://www.amd.com

[16] Hans Sagan, *Space-filling curves,* Springer-Verlag, 1994

[17] N. H. Chau, A. Kawai, T. Ebisuzaki, *Implementation of fast multipole algorithm on specialpurpose computer MDGRAPE-2*, Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics SCI2002, (Orlando, Colorado, USA, July 14-18, 2002), 477–481.

[18] N. H. Chau, A. Kawai, T. Ebisuzaki, Special-purpose computer accelerated fast multipole method, *Journal of Computer Science and Cybernetics*, 23 (3), (2007) 231-239.

[19] N. H. Chau, A. Kawai, T. Ebisuzaki, Acceleration of fast multipole method using special-purpose computer GRAPE, International *Journal of High Performance Computing Applications,* 22 (2)(2008)194-205.

[20] http://developer.nvidia.com/cuda-toolkit-40

[21] http://atlas.riken.jp

[22] http://www.riken.jp

Parallel GPU implementation of octree construction using space-filling curves

Nguyen Hai Chau

VNU University of Engineering and Technology, 144 Xuan Thuy, Hanoi, Vietnam

Octree is a hierarchical data structure that plays important roles in many research and application areas including molecular dynamics simulation (MDS). Construction of the octree is the first and an important step of fast force calculation algorithms in MDS such as fast multipole method (FMM), tree. In this paper, we implement the octree construction algorithm using Morton space-filling curve and parallelize the algorithm on nVidia's graphics processing units (GPU). Using the Morton space-filling curve and nVidia's GPUs, we have succeeded to speed up the octree construction for systems of 2^{20} to 10.2^{20} particles from 2.1 to 7 times. Experiment results show that main factors to the speed up of octree construction on GPU are data transfer speed between GPU and the host computer; and the speed of sorting particles on GPU by their Morton keys.