УДК 004.421.2:517.443

Р. Рикмас
Національний університет “Львівська політехніка”,
кафедра систем автоматизованого проектування

ПОРІВНЯННЯ ЕФЕКТИВНОСТІ РЕАЛІЗАЦІЙ ШПФ У ПРОГРАМНИХ БІБЛІОТЕКАХ

Наведено огляд реалізацій алгоритмів ШПФ (швидке перетворення Фур’є) в програмних бібліотеках, які поширені під час розроблення програмного забезпечення, зокрема FFTW, Intel IPP , CUDA CuFFT для реалізації ШПФ. Серед них визначено такі, які б сприяли максимально ефективному використанню обчислювальних ресурсів сучасного комп’ютера.

Ключові слова: швидке перетворення Фур’є, CUDA, FFTW, програмна бібліотека, графічний процесор.

This article provides an overview of implementations of algorithms FFT (Fast Fourier transform) in software libraries, which are widely used in software development. Analyzed the most widely used software libraries (FFTW, Intel IPP, CUDA CuFFT), which implemented FFT, including selected such that would make the most efficient use of computing resources of a modern computer are analyzed.

Keywords: fast Fourier transform, CUDA, FFTW, software library, graphic processor.

Вступ

Основою цифрової обробки даних є дискретні перетворення. Особливе місце серед них займає дискретне перетворення Фур’є. Широке застосування цифрової техніки та зростаючі вимоги до обробки ставлять особливі вимоги до алгоритмів їх обробки. Сьогодні ШПФ використовуються всюди, де потрібна обробка дискретних сигналів, зокрема під час спектрального аналізу, обробки зображень, відео, мови та звуку [1].

Сьогодні багатоядерні процесори чи навіть багатопроцесорні робочі станції є звичним явищем. Стрімко розвиваються нові стандарти для неграфічних обчислень зі застосуванням графічних процесорів (CUDA, OpenCL). Оскільки сучасний комп’ютер володіє великою кількістю різноманітних обчислювальних ресурсів, під час вибору конкретної бібліотеки ШПФ необхідно враховувати оптимальне використання всіх наявних ресурсів. Кількість бібліотек для обчислення ДПФ дуже велика і вони представляють різні мови програмування [2].

Найбільший інтерес викликають бібліотеки, реалізовані виробниками основних апаратних засобів комп’ютера: Intel, NVidia, оскільки вони повинні максимально ефективно використовувати свої апаратні ресурси [3, 4].

Огляд обраних реалізацій ШПФ

FFTW. Бібліотека у вигляді набору модулів мовами C та Fortran для обчислення швидкого перетворення Фур’є (ШПФ). FFTW дає змогу працювати як з дійсними, так і комплексними числами, з довільною кількістю вхідних даних, довжина яких не обов’язково кратна 2^n. Бібліотека також містить модулі, які дають змогу використовувати її на багатопроцесорних машинах із загальною та роздільною пам’яттю.

Робота з FFTW полягає в тому, що спочатку відбувається побудова «плану», який оптимізує час обчислення для заданої задачі. Потім побудований план передається як параметр функціям, які безпосередньо відповідають за обчислення ШПФ.

Для обчислення ШПФ бібліотека використовує багато алгоритмів, а вибір конкретного визначається за набором заданих параметрів та вхідних даних. Бібліотеку спроектовано так, що використовується генератор коду, який оптимально розбиває заданий об’єм даних на менші частини, які швидко обчислюються [2].

Особливість реалізації полягає в тому, що саме вона використовується в математичному пакеті MatLab [5].

Intel IPP. Intel IPP – це невелика бібліотека, яка спроектована для роботи з медіа-інформацією та обробки великої кількості даних. Бібліотека забезпечує підтримку багатоядерних процесорів та написана з використанням високоефективного коду на основі паралелізму інструкцій, який покладено в основу інструкцій MMX та SSE. Ці технології підвищують ефективність обчислень під час обробки сигналу, зображень та відео.

Бібліотека містить широкий набір функцій за дискретними перетвореннями: перетворення Фур'є, косинус-дискретне перетворення, перетворення Хартлі, Гільберта, Уолша-Адамара та вейвлет-перетворення сигналів.

У бібліотеці функції обчислення дискретного перетворення Фур'є розбиті на три групи, які відповідають використовуваним алгоритмам: Fast Fourier Transform Functions (алгоритм Кулі–Тьюкі), Discrete Fourier Transform Functions (алгоритм Винограда, та Prime-factor FFT algorithm) та Goertzel Functions (алгоритм Герцеля). За функціями ДПФ можна обчислювати пряме та зворотне перетворення над комплексними та дійсними числами [6].

CuFFT. Розроблена компанією NVIDIA та оптимізована на використання графічних процесорів (GeForce 8 та вище). Алгоритми, що реалізує бібліотека, реалізовані на основі технології CUDA – використанні графічних процесорів для неграфічних обчислень. CUFFT надає простий інтерфейс для паралельних обчислень ШПФ на NVIDIA GPU.

Бібліотека підтримує такі функції [7]:

- 1D, 2D і 3D перетворення комплексних та дійсних даних; результат перетворення можна записати у вхідний масив дійсних чи комплексних даних;
- Пакетне виконання декількох перетворень будь-якої розмірності паралельно;
- Перетворення масивів даних з 64 мільйонами елементів одинарної точності та 128 мільйонів елементів подвійної точності в будь-якій розмірності, не обов'язково кратній степені 2 (обсяг перетворення зазвичай обмежується кількістю доступної пам'яті GPU);
- Обчислення з подвійною точністю нових графічних процесорів (GT200 та новіші);
- Підтримка потокового виконання, яка дозволяє одночасне обчислення та переміщення даних;

Бібліотека реалізує оптимізовані перетворення за алгоритмом Кулі–Тьюкі за основою: 2, 3, 5 та 7. Отже, обчислення зводиться до розкладу числа за правилом $2^a \times 3^b \times 5^c \times 7^d$, де a, b, c, d – додатні цілі числа. Для всіх інших випадків використовується алгоритм Блюстрейна (Bluestein’s Algorithm for Arbitrary N) – як відомо, його ефективність поступово зменшується із збільшенням кількості вхідних даних [7].

Метод порівняння

Усі розглянуті бібліотеки використовують схожий інтерфейс та методи обчислення, навіть при написанні коду відмінності полягають лише в назвах викликаних функцій.

Для тестування можна виділити три категорії за кількістю вхідних даних:

1. Числа, кратні степеню 2 (алгоритм Кулі–Тьюкі);
2. Прості числа (алгоритм Блюстрейна);
3. Числа, які можна розкласти за правилом: $2^a \times 3^b \times 5^c \times 7^d$.

Оскільки реалізації бібліотек різні та невідомі і оцінити продуктивність обчислення не можливо, то порівняння можна здійснити на основі тривалості обчислення.

Вхідним параметром для обчислення завжди є кількість елементів послідовності, над якою необхідно здійснити перетворення. Всі обчислення проводити можна з числами типу double для забезпечення високої точності результату.

Код для бібліотеки FFTW:

```
int i, test;
const int N_FFTS = 8;

fftw_init_threads();
fftw_plan_with_nthreads(N_THREADS);

fftw_complex *in, *out;
in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * size);
out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * size);

fftw_plan p;
p = fftw_plan_dft_1d(size, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
for( i = 0; i < size; i++ ){
      in[i][0]=i;
      in[i][1]=2.0*i;
}

for( test = 0; test < N_FFTS; test++ ){
      fftw_execute(p);
}

fftw_destroy_plan(p);
fftw_free(in); fftw_free(out);
fftw_cleanup_threads();
```

Код для бібліотеки IPP:

```
int i, test;
const int N_FFTS = 8;

Ipp64fc *in, *out;
in = (Ipp64fc*) ippsMalloc_64fc(sizeof(Ipp64fc) * size);
out = (Ipp64fc*) ippsMalloc_64fc(sizeof(Ipp64fc) * size);
for( i = 0; i < size; i++ ){
      in[i].re = i;
      in[i].im = 2.0*i;
}

IppsDFTSpec_C_64fc* p;
ippsDFTInitAlloc_C_64fc(&p, size, IPP_FFT_DIV_BY_SQRTN, ippAlgHintFast);

for(test = 0; test < N_FFTS; test++){
      ippsDFTFwd_CToC_64fc(in, out, p, NULL);
}

ippsDFTFree_C_64fc(p);
ippsFree(in); ippsFree(out);
```

Код для бібліотеки cuFFT:

```
int i, test;
const int N_FFTS = 8;
LARGE_INTEGER t_init, t_stop;
LARGE_INTEGER temp;
QueryPerformanceFrequency((LARGE_INTEGER*) &temp);
double freq = ((double) temp.QuadPart) / 1000.0;
cuDoubleComplex *inMemory = (cuDoubleComplex *) malloc(
sizeof(cuDoubleComplex) * size);
cuDoubleComplex *outMemory = (cuDoubleComplex *) malloc(
sizeof(cuDoubleComplex) * size);
for( i = 0; i < size; ++i ){
      inMemory[i].x = i;
      inMemory[i].y = i*2.0;
}
```

```
cuDoubleComplex *in = NULL;
cuDoubleComplex *out = NULL;
cutilSafeCall(cudaMalloc((void**)&in,sizeof(cuDoubleComplex)*size));
cutilSafeCall(cudaMalloc((void**)&out,sizeof(cuDoubleComplex)*size));
cudaMemcpy(in, inMemory, sizeof(cuDoubleComplex)*size,
cudaMemcpyHostToDevice);

cufftHandle plan;
cufftResult result = cufftPlan1d(&plan,size,CUFFT_Z2Z,1);

QueryPerformanceCounter((LARGE_INTEGER*) &t_init);
for(test = 0; test < N_FFTS; test++){
      result = cufftExecZ2Z(plan,in,out,CUFFT_FORWARD);
}
QueryPerformanceCounter((LARGE_INTEGER*) &t_stop);
cudaMemcpy(outMemory, out, sizeof(cuDoubleComplex)*size,
cudaMemcpyDeviceToHost);

double duration = (((double) t_stop.QuadPart – (double) t_init.QuadPart) /
freq) / N_FFTS;

cufftDestroy(plan);
cudaFree(in); cudaFree(out);
free(inMemory);free(outMemory);
```

Для підвищення продуктивності обчислення за допомогою центрального процесора необхідно ознайомитись з додатковими технологіями, які впливають на результати роботи, зокрема з технологією паралельного обчислення OpenMP, набором команд мікропроцесора AVX.

OpenMP. Технологія OpenMP дає змогу створювати багатопоточні програми для комп'ютерів з багатоядерними процесорами за допомогою директив компілятора [1]. У випадку з бібліотеками FFTW та IPP можна передбачити багатопоточне виконання ШПФ, реалізоване за допомогою технології OpenMP.

AVX. Під час роботи з бібліотеками FFTW та IPP можна використати новий набір команд процесорів – AVX (Advanced Vector Extensions). У результаті використання нового набору команд тестуванням виявлено підвищення продуктивності обчислень на понад 15 %.

Аналіз результатів

Результати обчислень для порівняння зручно подати у вигляді графіків, за яким видно залежність тривалості обчислень від кількості вхідних даних для кожної з послідовностей. Тестування проводилось на комп'ютері з центральним мікропроцесором Intel Core i7-2600@3.4 та графічним процесором GF108 на NVIDIA Quadro 600.

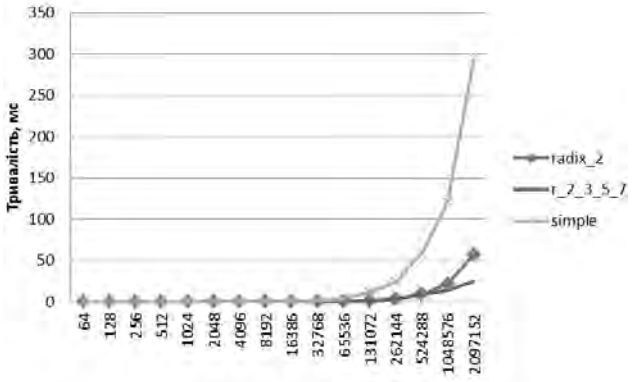

Рис. 1. Тривалість обчислення із використанням FFTW

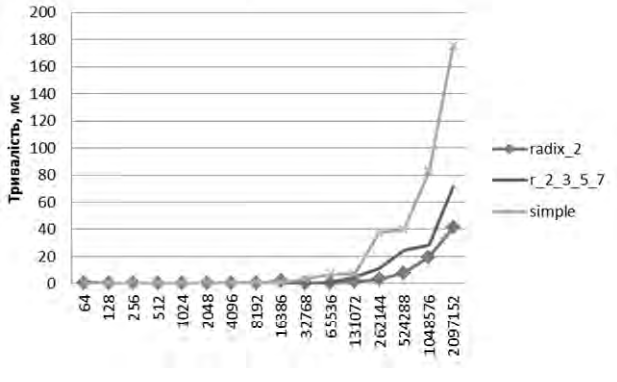

Рис. 2. Тривалість обчислення із використанням IPP

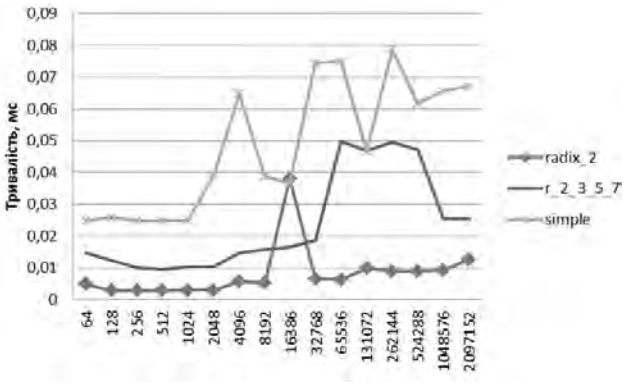

Рис. 3. Тривалість обчислення із використанням CUDA

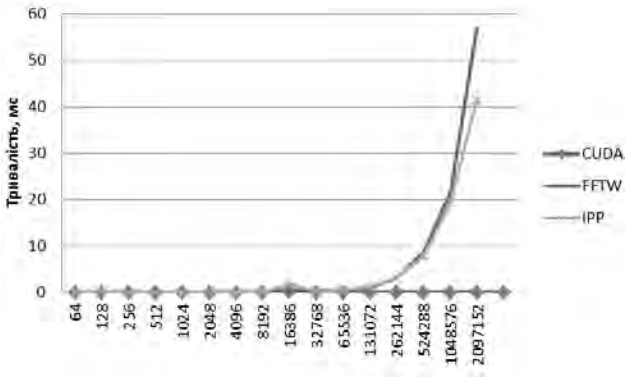

Рис. 4. Спільне порівняння бібліотек під час роботи з послідовностями даних, кратних цілому степеню 2

Висновки

У процесі роботи з бібліотеками виявлено загальні тенденції до розбиття великого набору даних на малі частини, до яких можна застосувати ефективніші алгоритми Винограда, та для випадків, коли розклад не можливий, – алгоритм Блюстрейна.

Порівнянням бібліотек для обчислення ШПФ виявлено, що всі програмні бібліотеки дають змогу здійснювати перетворення великої кількості даних з високою швидкістю, отже, їх можна використовувати навіть для обчислень у реальному часі. Проте існує значний розрив у продуктивності обчислень для послідовностей довжиною, кратною цілому степеню числа 2 та довжиною, яка дорівнює простому числу. Важливою є робота із розроблення нових ефективних алгоритмів для обчислення ШПФ [9].

Також впевнено можна стверджувати, що значну перевагу в обчисленні ШПФ мають графічні процесори, а наявність стандарту OpenCL [3, 8] робить написання бібліотек для обчислення ШПФ з використанням графічних процесорів дуже перспективним.

1. [Електронний ресурс]. – Режим доступу. http://uk.wikipedia.org/. 2. [Електронний ресурс]. – Режим доступу. http://www.fftw.org/. 3. [Електронний ресурс]. – Режим доступу. http://developer. nvidia.com/cufft/. 4. [Електронний ресурс]. – Режим доступу. http://software.intel. com/en-us/articles/intel-ipp/. 5. [Електронний ресурс]. – Режим доступу. http://www.mathworks.com. 6. Integrated Performance Primitives 7.0 Documentation. 7. CUDA CUFFT Library Documentation. 8. [Електронний ресурс]. – Режим доступу. http://software.intel.com/en-us/articles/opencl-sdk/9999 9. Процько І.О. Обчислювальні структури адаптивного до обсягу ШПФ // Вісник Нац. ун-ту "Львівська політехніка". № 651. – 2009. – С. 145–150.