Praca z buforem liniowym - przetwarzanie łańcucha znaków

Akceleracja Algorytmów Wizyjnych

Jan Rosa 20.03.2025

Kod kernela

```
__kernel void helloworld(__global char* in, __global char* out_1, __global char*
out_2)
{
	int num = get_global_id(0);
	if (in[num] >= 65 && in[num]<=90)
	{
		out_1[num] = in[num] + (97-65);
	}
	else if (in[num] >= 97 && in[num]<=122)
	{
		out_1[num] = in[num] - (97-65);
	}
	else
	{
		out_1[num] = in[num];
	}


	printf("%d\n\r", num);
	switch (in[num]) {
		case '0':		out_2[4*num] = '0'; out_2[4*num+1] = '0';
out_2[4*num+2] = '0'; out_2[4*num+3] = '0'; break;
		case '1':		out_2[4*num] = '0'; out_2[4*num+1] = '0';
out_2[4*num+2] = '0'; out_2[4*num+3] = '1'; break;
		case '2':		out_2[4*num] = '0'; out_2[4*num+1] = '0';
out_2[4*num+2] = '1'; out_2[4*num+3] = '0'; break;
		case '3':		out_2[4*num] = '0'; out_2[4*num+1] = '0';
out_2[4*num+2] = '1'; out_2[4*num+3] = '1'; break;
		case '4':		out_2[4*num] = '0'; out_2[4*num+1] = '1';
out_2[4*num+2] = '0'; out_2[4*num+3] = '0'; break;
		case '5':		out_2[4*num] = '0'; out_2[4*num+1] = '1';
out_2[4*num+2] = '0'; out_2[4*num+3] = '1'; break;
		case '6':		out_2[4*num] = '0'; out_2[4*num+1] = '1';
out_2[4*num+2] = '1'; out_2[4*num+3] = '0'; break;
		case '7':		out_2[4*num] = '0'; out_2[4*num+1] = '1';
out_2[4*num+2] = '1'; out_2[4*num+3] = '1'; break;
		case '8':		out_2[4*num] = '1'; out_2[4*num+1] = '0';
out_2[4*num+2] = '0'; out_2[4*num+3] = '0'; break;
		case '9':		out_2[4*num] = '1'; out_2[4*num+1] = '0';
out_2[4*num+2] = '0'; out_2[4*num+3] = '1'; break;
		case 'A': case 'a': out_2[4*num] = '1'; out_2[4*num+1] = '0';
out_2[4*num+2] = '1'; out_2[4*num+3] = '0'; break;
		case 'B': case 'b': out_2[4*num] = '1'; out_2[4*num+1] = '0';
out_2[4*num+2] = '1'; out_2[4*num+3] = '1'; break;
		case 'C': case 'c': out_2[4*num] = '1'; out_2[4*num+1] = '1';
out_2[4*num+2] = '0'; out_2[4*num+3] = '0'; break;
		case 'D': case 'd': out_2[4*num] = '1'; out_2[4*num+1] = '1';
out_2[4*num+2] = '0'; out_2[4*num+3] = '1'; break;
```

```
            case 'E': case 'e': out_2[4*num] = '1'; out_2[4*num+1] = '1';
out_2[4*num+2] = '1'; out_2[4*num+3] = '0'; break;
            case 'F': case 'f': out_2[4*num] = '1'; out_2[4*num+1] = '1';
out_2[4*num+2] = '1'; out_2[4*num+3] = '1'; break;
            default:                out_2[4*num] = '_'; out_2[4*num+1] = '_';
out_2[4*num+2] = '_'; out_2[4*num+3] = '_'; break;
      }

}
```

Fragmenty Kodu Aplikacji sterującej

Kod inicjalizujący bufory i obsługujący wejścia tekstowe

```
      /*Step 7: Initial input,output for the host and create memory objects for
the kernel*/
      char input[0xFFF] = {0};
      if (argc > 1)
      {
            strcpy(input, (const char*)argv[1]);
      }
      else
      {
            cout << "Write input string:" << endl;
            cin >> input;
      }
      cout << "Your input string is:" << endl;
      cout << input << endl;
      size_t strlength = strlen(input);
      cout << "Length of input is:" << strlength << endl;
      size_t out_1_strlength = strlen(input);
      char *output_1 = (char*) malloc(out_1_strlength);
      size_t out_2_strlength = 4*strlen(input);
      char *output_2 = (char*) malloc(out_2_strlength);

      cl_mem inputBuffer = clCreateBuffer(context, CL_MEM_READ_ONLY|
CL_MEM_COPY_HOST_PTR, (strlength + 1) * sizeof(char),(void *) input, NULL);
      cl_mem outputBuffer_1 = clCreateBuffer(context, CL_MEM_WRITE_ONLY ,
out_1_strlength * sizeof(char), NULL, NULL);
      cl_mem outputBuffer_2 = clCreateBuffer(context, CL_MEM_WRITE_ONLY ,
out_2_strlength * sizeof(char), NULL, NULL);
      cout << "Length of output is:" << out_1_strlength << endl;
```

Dodanie do kernela nowych argumentów.

```
      status = clSetKernelArg(kernel, 1, sizeof(cl_mem), (void
*)&outputBuffer_1);
      status = clSetKernelArg(kernel, 2, sizeof(cl_mem), (void
*)&outputBuffer_2);
```

Przykładowe uruchomienie

```
No GPU device available.
Choose CPU as default device.
Your input string is:
qAqBcd1234567
Length of input is:13
Length of output is:13

output_1 string:
QaQbCD1234567

output_2 string:
____1010____10111100110100010010001101000101011001111
Passed!
```

Ocena zasadność implementacji operacji na łańcuchach w GPU

Operacje na łańcuchach w GPU są często nieefektywne ze względu na narzut transferu danych i brak naturalnej równoległości. Konwersja wielkich liter na małe czy zamiana liczb szesnastkowych na binarne powodują rozgałęzienia kodu, co spowalnia działanie. CPU z SIMD radzi sobie lepiej. GPU sprawdza się natomiast w równoległym wyszukiwaniu wzorców w dużych zbiorach danych, np. analizie logów czy genomów, gdzie wiele bloków może niezależnie przetwarzać fragmenty tekstu, minimalizując problem synchronizacji i efektywnie wykorzystując moc obliczeniową kart graficznych.