

Комплекс средств программирования реконфигурируемых вычислительных систем на основе ПЛИС



А.И. Дордопуло,



НИЦ СЭ и НК

И.И. Левин, В.А. Гудков, А.А. Гуленок,
А.В. Бовкун, Г.А. Евстафьев, К.Н. Алексеев

Компилятор программ на языке C в программы на COLAMO с последующей трансляцией в VHDL

Преобразовать последовательную программу на языке программирования C (стандарт ISO/IEC 9899:1999 для компилятора gcc) в информационный граф, описанный на языке программирования COLAMO, который транслятором COLAMO транслируется в конфигурационные файлы ПЛИС для заданного вычислительного ресурса.

Выходные данные: конфигурационные файлы кристаллов ПЛИС с параллельно-конвейерной реализацией алгоритма программы на специализированном вычислителе потоковой архитектуры.

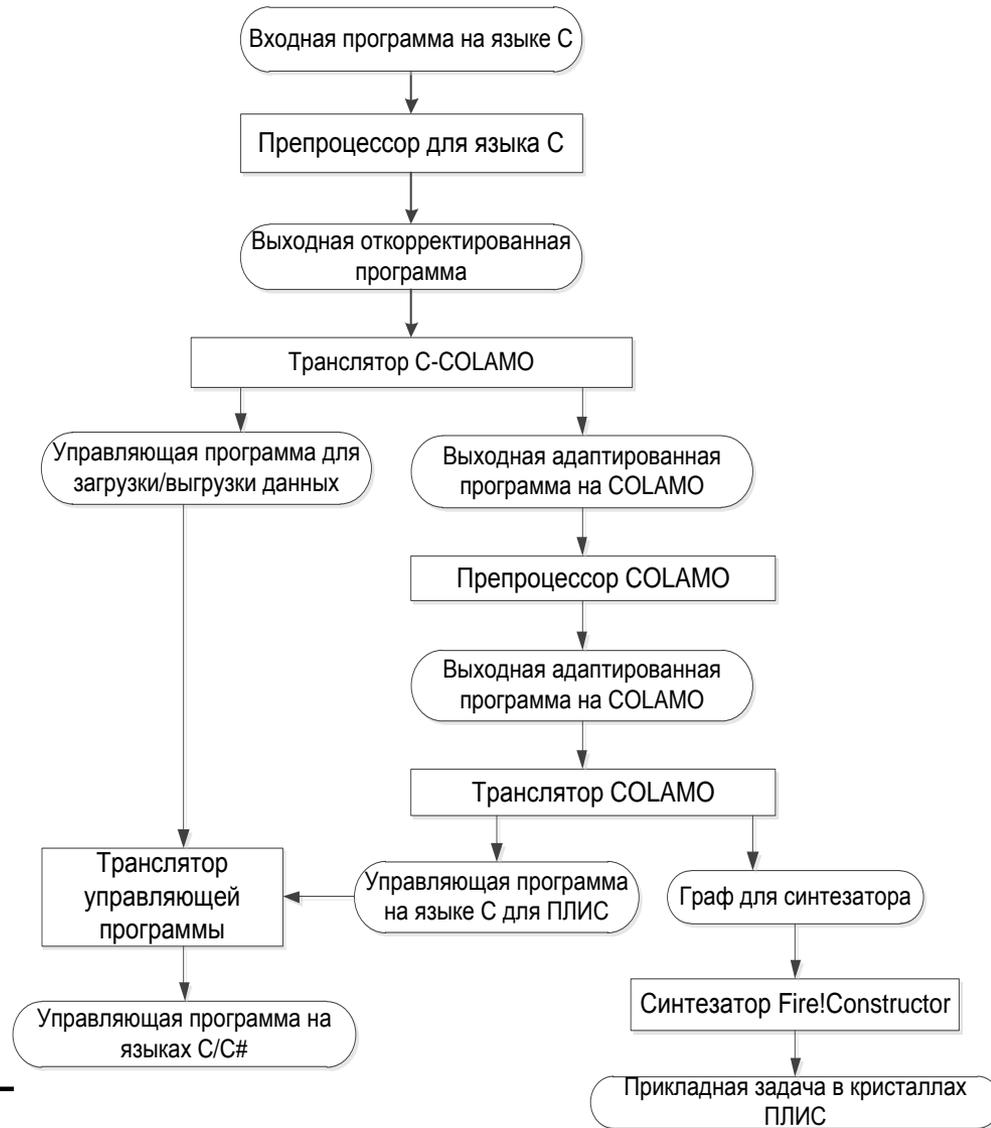
Заданные показатели: удельная производительность получаемых решений не менее 70% при уровне заполнения кристалла не выше 90%.

Аналоги: Vivado HLS, Quartus HLS, Mitrion-C, Catapult-C

Отличия: создание решения задачи на выбранной PBC, а не IP-ядра₂

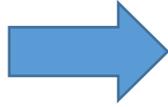
Общая схема предлагаемого решения

1. Преобразование входной программы к каноническому виду
2. Построение абсолютно-параллельного информационного графа входной программы, отображение его в синтаксис COLAMO.
3. Преобразование абсолютно-параллельного информационного графа входной программы к параллельной, параллельно-конвейерной или конвейерной форме.
4. Адаптация программы к вычислительному ресурсу.
5. Трансляция программы в VHDL и синтез конфигурационных файлов.

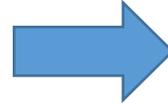


Структура программного комплекса

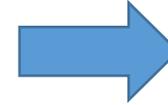
Транслятор
«Ангел»



Процессор
«Русалка»



Процессор
«Прокруст»



Процессор
«Щелкунчик»



Преобразует последовательную программу на С в информационный граф на COLAMO

Преобразует абсолютно параллельную программу на COLAMO в параллельно-конвейерную ресурсонезависимую форму

Автоматически подбирает параметры параллельно-конвейерной программы на COLAMO для заданного вычислительного ресурса и преобразует программу для эффективной реализации

Редуцирует программу при нехватке аппаратного ресурса для ее эффективной реализации на имеющемся

Транслятор языка COLAMO + синтезатор Fire!Constructor + описание архитектуры PBC



Синтезатор загрузочных конфигурационных файлов Xilinx Vivado/ISE



PBC: загрузочные конфигурационные файлы ПЛИС



Транслятор «Ангел»

1. Преобразует входную программу на языке С к каноническому виду
2. Строит абсолютно-параллельный информационный граф входной программы и отображает его в синтаксис языка COLAMO.

Состоит из 2 модулей: препроцессора языка С и транслятора С-COLAMO.

Модуль препроцессора языка С:

- проверяет синтаксическую правильность входной программы на С;
- раскрывает директивы условной компиляции и макросы, преобразует трехместные операции к двуместным, преобразует SWITCH и CASE к оператору IF, приводит все циклы к оператору FOR, преобразует доступ по указателям в адресацию массивов;
- проверяет синтаксическую правильность преобразованной программы на С.



Транслятор «Ангел»

Модуль транслятора C-COLAMO:

- выполняет анализ информационных зависимостей переменных и массивов исходной последовательной программы;
- на основе результатов анализа строит абсолютно-параллельный информационный граф;
- преобразует последовательную программу с произвольным обращением к памяти к потоку данных;
- определяет характерные для последовательных программ нарушения правил однократного присваивания и единственной подстановки;
- устраняет нарушения правил в тексте программы на COLAMO (расщепляет скалярные переменные и растягивает массивы по итерациям с копированием необходимых данных);
- для каждой индексной переменной каждого массива определяет тип цикла (итерационный или слоевой) для всех вхождений переменной в тексте программы;

Выходные данные: абсолютно параллельная программа в синтаксисе COLAMO с большим числом каналов КРП.

Процессор «Русалка»



Преобразует абсолютно-параллельный информационный графа входной программы на COLAMO к параллельной, параллельно-конвейерной или конвейерной форме:

- выполняет разбиение каждой размерности массивов данных на параллельную (Vector) и потоковую (Stream) составляющую;
- все циклы в программе на COLAMO разбивает на параллельную (Vector) и потоковую (Stream) составляющую;
- выполняет вынос зависящих только от итераций переменных из слоевых циклов;
- определяет и преобразует базовый подграф вычислений в вычислительную структуру LET или подкадр (SubCadr);
- преобразует базовый подграф в структуру Implicit;
- вносит необходимые корректировки в текст программы на COLAMO с учетом сделанных изменений;

Выходные данные: параметризуемая по параллельной и потоковой составляющим параллельно-конвейерная программа на языке COLAMO, информационно-эквивалентная входной абсолютно-параллельной программе.

Процессор «Прокруст»



1. Автоматически рассчитывает параметры параллельно-конвейерной программы на COLAMO для заданного вычислительного ресурса;
2. Преобразует программу для эффективной реализации на заданном вычислительном ресурсе:

- выполняет подбор параметров ресурсонезависимой программы в зависимости от критического ресурса: время обработки, число КРП, доступный аппаратный ресурс;
- для рассчитанных параметров преобразует входную ресурсонезависимую программу к параллельной, конвейерной или параллельно-конвейерной форме;
- преобразует зависящие от итераций переменные к типам Intermet или Reg;
- выполняет организацию и синхронизацию обратных связей для реализуемой формы организации вычислений;
- синтезирует многокадровые программы для структур SubCadr и Let;
- организует вычисления по схеме «конвейер в конвейере».

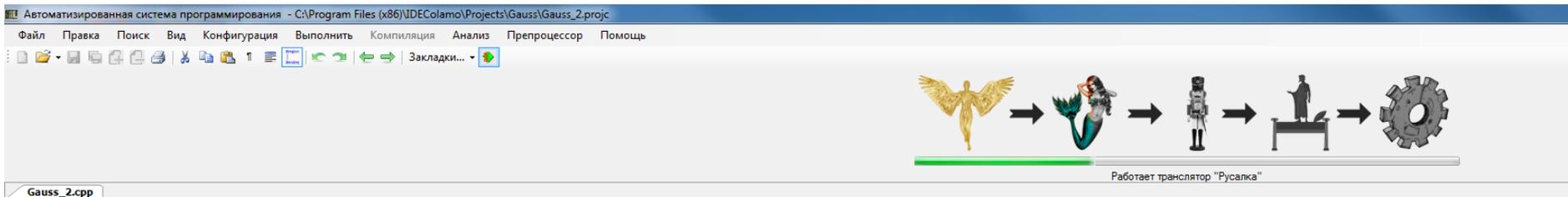


Процессор «Щелкунчик»

1. Редуцирует программу при нехватке аппаратного ресурса для реализации базового подграфа вычислений (если «Прокруст» не может подобрать параметры даже для минимального уровня параллелизма);
2. Преобразует программу на COLAMO для эффективной реализации на имеющемся вычислительном ресурсе:
 - рассчитывает требуемый коэффициент сокращения аппаратных затрат задачи;
 - выполняет редукцию производительности базового подграфа по числу подграфов, вычислительных операций и разрядности;
 - выбирает наиболее рациональный вариант реализации задачи по критерию общего времени обработки.
 - преобразует текст программы на COLAMO.

Выходные данные: редуцированная программа в синтаксисе COLAMO, которая может быть выполнена на имеющемся вычислительном ресурсе.

Пример работы комплекса при трансляции задачи решения СЛАУ методом Гаусса



```
Gauss_2.cpp
64     for(j = 0; j < N + 1; j++)
65     {
66         fscanf(file,"%d\n", &X[i][j]);
67     }
68 }
69 fclose(file);
70 }
71 }
72
73
74 int main()
75 {
76     int i , j , k, t, r;
77     float d;
78
79     char input_file_name[80]="input_data.txt";
80     char output_file_name[80]="output_data.txt";
81     load_data(input_file_name, (float**)matrix);
82     //return 1;
83
84     //////////////////////////////////////
85     for(i = 0 ; i < N-1 ; i++){
86         for(j = i+1 ; j < N ; j++)
87         {
88             d = (matrix[j][i]/matrix[i][i]);
89
90             for(k = i; k < N+1 ; k++)
91                 matrix[j][k]-=(matrix[i][k]*d);
92         }
93     }
94
95     save_dataMatrix(output_file_name, (float**)matrix);
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Конфигурация

Библиотеки Средства трансляции Средства синтеза Настройки оболочки Проект

Интерфейсная библиотека KUS_Tertius Генерация потокового компонента на языке Argus

Операционная библиотека US_Lib Синхронизация контроллеров распределенной памяти

Среда проектирования Vivado Генерировать управляющий компонент C++

Тактовая частота 850 Использовать битовую обработку

Вычислительных модулей 1 Использовать синхронизатор обратной связи

Интерфейс Поточковый Размещать итоговые файлы проекта в отдельном каталоге

Макс. время отклика конвейера 3 Сек Удалять промежуточные данные

Режим эксперта Искать изоморфные структуры

Экспертные настройки

КРП 2

Козф. использования ресурсов 100 %

Отмена Применить и закрыть

Системные сообщения

№	Компонент	Сообщение
24	Colamo	Время выполнения синтаксического анализа 0 ч. 0 мин. 0 сек. (46 микросек.).
25	Colamo	Выполняется компиляция проекта
26	Colamo	Время трансляции проекта 0 ч. 0 мин. 0 сек. (308 микросек.).
27	Colamo	Трансляция проекта Gauss_2_mod выполнена успешно

Пример работы комплекса при трансляции задачи решения СЛАУ методом Гаусса

The screenshot displays the FireGUI interface for a Verilog HDL project named 'Gauss_2_mod.gps'. The main window shows a complex logic circuit diagram with various functional blocks and interconnections. The blocks are organized into several columns, representing different stages of the Gaussian elimination process. Key blocks include:

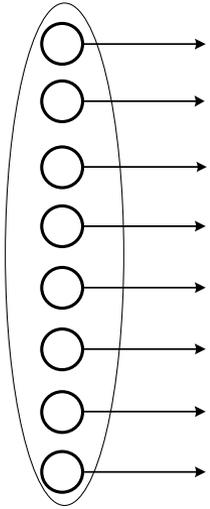
- ms2xddd vnd** (EL55_mx2vdd4_ID564): Multiplier blocks.
- dec_ddd vnd** (EL71_dec_ID658, EL135_dec_ID936, EL138_dec_ID944, EL501_and_ID2056): Decision and comparison blocks.
- dpram_ddd164 vnd** (EL5_dpram_ID475, EL6_dpram_ID476): Memory blocks.
- comp_eq_fsv_ddd_rg_vnd** (EL136_comp_ID941, EL137_comp_ID1220, EL139_comp_ID949, EL140_comp_ID1224, EL141_comp_ID1224, EL142_comp_ID1224, EL143_comp_ID1224, EL144_comp_ID1224, EL145_comp_ID1224, EL146_comp_ID1224, EL147_comp_ID1224, EL148_comp_ID1224, EL149_comp_ID1224, EL150_comp_ID1224): Comparison blocks for equations.
- and_1** (EL216_and_ID1209, EL442_and_ID1997, EL471_and_ID2026, EL137_and_ID943, EL218_and_ID1224, EL443_and_ID1998, EL367_and_ID1730, EL472_and_ID2027): AND logic blocks.
- div_fp_32n** (EL278_div_fp_32n_ID1576, EL281_div_fp_32n_ID1580): 32-bit floating-point dividers.
- mul_fp_32n_1dsp** (EL279_mul_ID1578, EL282_mul_ID1582): 32-bit floating-point multipliers.
- sub_fp_32n_log** (EL280_sub_ID1579): 32-bit floating-point subtractor.
- comp_ge_fsv_ddd_rg_vnd** (EL369_comp_ID1738, EL372_comp_ID1748): Comparison blocks for greater-than-or-equal-to.
- comp_ge_fsv_ddd_rg_vnd** (F1368_comp_ID1734): Comparison block for greater-than-or-equal-to.

The status bar at the bottom indicates 35 warnings and 1 critical warning. The console window shows the following output:

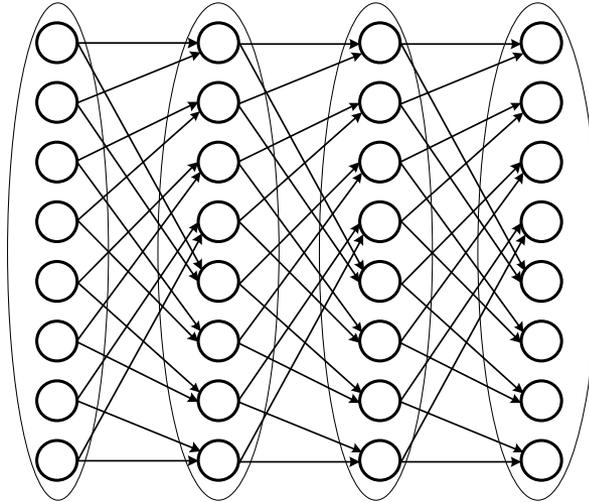
```
VHDLInclude закончил работу: 0ч. 0 мин. 15 сек. (74 миллисек.)
Файл C:\Program Files (x86)\IDEColano\Projects\Gauss\ves\TranslatorColano\EP\Generation\Gauss_2_mod\ResultFireConstructor.xml успешно создан
Конец 11:49:12
Прошло времени: 0ч. 0 мин. 17 сек. (401 миллисек.)
-> Обнаружено <0> ошибок(ок)=-
```

Виды прикладных задач

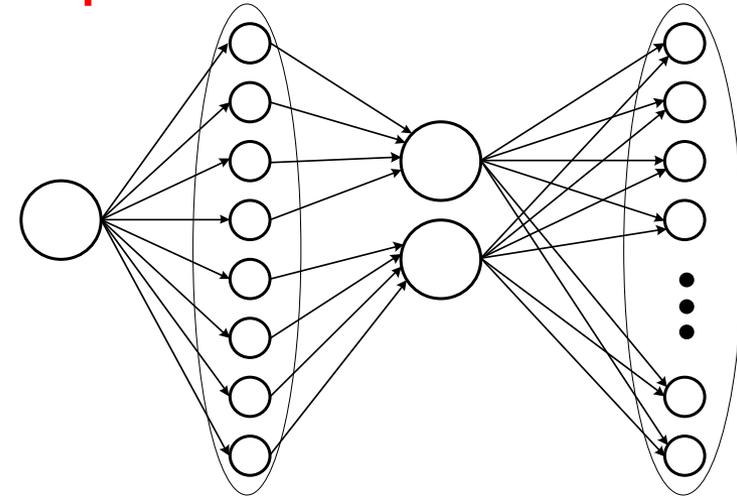
Линейные



Потоковые с постоянной интенсивностью



Потоковые с переменной интенсивностью



- **Линейные** - символьная обработка;
- **Потоковые** – линейная алгебра (решение СЛАУ методом Гаусса, решение СЛАУ методом Якоби для полных и 3-диагональных матриц, LU-разложение);
- С переменной интенсивностью потоков данных - докинг ингибиторов, молекулярное моделирование и др.

Сравнение реализации задачи символьной обработки на COLAMO, Vivado HLS и транслятором C-COLAMO

Для микросхемы Xilinx Virtex 7 XC7VX485T (РВС «Тайгета»).



Частота 250 МГц. САПР Xilinx Vivado 2018.2

Наименование ресурса	Реализация программиста на COLAMO (двойной конвейер)	Автоматическая реализация программы C в Vivado HLS	Реализация программы C в трансляторе C-COLAMO	Эффективность C-Colamo	
				К COLAMO (один конвейер)	К Vivado HLS
Slice Registers (всего 607,200)	3 475	12 545	2 152	0,81	5,83
Slice LUTs (всего 303,600)	2 336	8 586	1 493	0,78	5,75
Occupied Slices (всего 75,900)	627	2 371	394	0,8	6,02
Максимальное число конвейеров в ПЛИС	120*2=240	30	150/195 (130%)	0,81 (195/240)	6,5 (195/30)
Время трансляции проекта в Vivado	12 ч. 10 мин.	7 ч. 24 мин.	15 ч. 34 мин.		

Сравнение реализации задачи символьной обработки на COLAMO, Vivado HLS и транслятором C-COLAMO

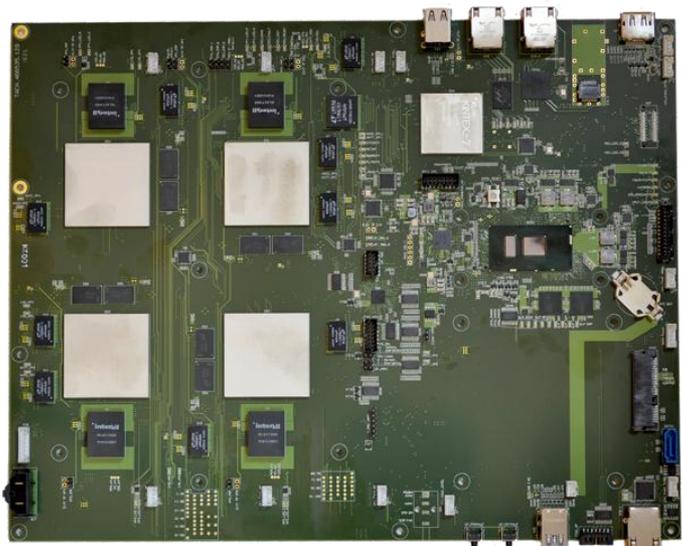
Для микросхемы Xilinx Kintex UltraScale
ХСКU095 (РВС «Неккар»)
Частота 400 МГц. САПР Xilinx Vivado
2018.2



Наименование ресурса	Реализация программиста на COLAMO	Реализация программы C в Vivado HLS	Автоматическая реализация программы C в трансляторе C-COLAMO	Эффективность C-Colamo	
				К COLAMO (один конвейер)	К Vivado HLS
CLB Registers (всего 1,075,200)	2 900	16 512	2 921	0,5	5,65
CLB LUTs (всего 537,600)	2 076	1 863	1 518	0,68	1,23
CLB (всего 67,200)	336	1 101	210	0,8	5,24
Максимальное число конвейеров в ПЛИС	205*2=410	60*	266/313 (118%)	0,76 (313/410)	5,21 (313/60)
Время трансляции проекта в Vivado	6 ч. 11 мин.	6 ч. 21 мин.	6 ч. 28 мин.		

*в результате разводки 60 конвейеров, полученных с помощью Vivado HLS (плотное заполнение), САПР Xilinx Vivado HLS выдал 54037 временных ошибок. Для 50 конвейеров - 30743 временные ошибки.

Результаты реализации задач линейной алгебры на НРК «Терциус»



Микросхема Xilinx
Virtex UltraScale
XCKU095T



Частота 250 МГц.
САПР Vivado 2018.2

Алгоритм	Число ступеней на плате (4 ПЛИС)	
	Автоматический расчет (100% ресурса)	Vivado HLS
Гаусс 8000*8000	821	1*
LU 8000*8000	767	1*
Якоби 3-д 8000*8000	950	1*

Компилятор программ на языке C в программы на COLAMO с последующей трансляцией в VHDL

Основные отличия от существующих трансляторов C-VHDL и Vivado HLS:

1. Входной текст на «чистом» C в стандарте ISO/IEC 9899:1999.
2. Создание масштабируемой как по данным, так и по итерациям параллельно-конвейерной программы.
3. Автоматическая адаптация программы под доступный аппаратный ресурс.
4. Поддержка многокристалльных проектов.
5. Автоматический синтез управляющей программы для регистрового интерфейса.

Спасибо за внимание!

Если остались вопросы:

dordopulo@superevm.ru

Текущие требования к исходному тексту программ на языке C

1. Текст программы не должен содержать следующих операторов и конструкций языка C: операторов **goto**, конструкций **union** и **struct**, функций **strlen** и функций работы с памятью (**calloc**, **memcpy** и др).

2. В тексте программы должны обязательно присутствовать процедуры загрузки и сохранения данных с фиксированными именами (**load_data**, **load_data32**, **load_data64** и **load_data_str**) для определения входных и выходных переменных.

3. В циклах должны использоваться обычные индексные переменные без косвенной адресации:

Неправильно: `for(ind[LenP_]=0 ; ind[LenP_] < AlphPower[LenP_] ; ind[LenP_]++)`

Правильно: `for(i=0 ; i < AlphPower[LenP_] ; i++)`

4. По возможности избегать неоправданно частого использования операторов **break** и **return** для немедленного (аварийного) выхода из циклов.

Общая методика преобразования последовательной программы на С в параллельную программу на COLAMO

1 этап (Ангел): перевод программы на языке программирования С в абсолютно-параллельную форму на языке программирования COLAMO

1. Все инициализируемые на С массивы (для чтения или записи данных) становятся мемориальными массивами в COLAMO с параллельным типом доступа (векторными по всем измерениям):

<code>int M [R][C]</code>	<code>Var M : Array Integer [R: Vector, C: Vector] Mem;</code>
---------------------------	--

2. Для каждого мемориального массива в COLAMO объявляется коммутационный массив-копия для использования в базовом подграфе вычислений:

<code>int M [R][C]</code>	<code>Var M : Array Integer [R: Vector, C: Vector] Mem;</code>
---------------------------	--

	<code>Var MCom : Array Integer [R:Vector,C:Vector] Com;</code>
--	--

3. Коммутационные массивы-копии источников данных инициализируются значениями мемориальных массивов. В тексте программы все мемориальные массивы заменяются на коммутационные массивы-копии.

	<code>MCom[*,*] := M [*,*];</code>
--	------------------------------------

Общая методика преобразования

2 этап (Ангел): анализ информационных зависимостей исходной программы

1. Поиск явной и неявной информационных зависимостей массивов данных и определение вида цикла (слоевой или итерационный) для каждой индексной переменной :

Явная	Неявная (псевдоитерационная)
$x[i] := f(x[i-k])$	$x[i] := f(x[i])$ $a := f(a)$

2. Критерии определения вида цикла для индексной переменной.

2.1. По умолчанию все циклы абсолютно-параллельной программы считаются итерационными;

2.2. Цикл для каждого измерения массива считается слоевым, если :

- в массив, индексируемый переменной цикла, в теле цикла производится только запись (без операций чтения);
- если в массив, индексируемый переменной цикла, в теле цикла производится и чтение, и запись при неявно заданной информационной зависимости.

2.3. Если в массив, индексируемый переменной цикла, в теле цикла производится и чтение, и запись, при явно заданной информационной зависимости цикл остается итерационным.

Общая методика преобразования

2 этап(Ангел): расщепление скалярных переменных и растягивание массивов по итерациям

1. Растягивание коммутационного массива-копии по итерациям:

```
for i = 1 to n do
  for j = 1 to m do
    x[j] = a*x[j-1]+
      y[i];
```

```
Var xCom : Array Integer [n: Vector, n+1:
Vector] Com;
...
xCom[0,1] = x[0];
for i = 1 to n do
  begin
    for j = 1 to m do
      xCom[j,i] = a*xCom[j-1,i] + y[i];
    // переписывание данных на следующую
итерацию по i
      xCom[* , i+1] = xCom[* , i];
  end;
```

2. Разрешение правила однократного присваивания:

```
for i = 1 to n do
  for j = 1 to m do
    x[j] = a*x[j-1]+
      y[i];
```

```
xCom[* ,1, 0] = x [*];
for i = 1 to n do
  begin
    for j = 1 to m do
      xCom[j,i,1]=a*xCom[j-1,i,0] + y[i];

      xCom[* ,i+1,0] = xCom[* ,i,1];
  end;
```

Общая методика преобразования

3 этап(Русалка): переход в параметризуемую параллельно-конвейерную форму

1. **Каждое** измерение всех мемориальных и коммутационных массивов разбивается на 2 составляющие – **векторную** (с параллельным) и **потокową** (с последовательным) типами доступа. Длина потоковой составляющей по умолчанию задается как $R = 1$, а длина векторной – как (исходная длина измерения массива $+R-1$)/ R).
2. Все циклы программы также разбиваются векторную и потоковую составляющие.
3. Подготовка входных данных для масштабирования программы процессором «Прокруст».

```
Var xCom : Array Integer
[n: Vector, n+1: Vector] Com;
...
for i = 1 to n do
  for j = 1 to m do
    xCom[j,i] =
      a*xCom[j-1,i]+y[i];
```

```
Var xCom : Array Integer
[(n+R1-1)/R1: Vector, R1: Stream,
(n+1+R2-1)/R2: Vector, R2: Stream] Com;
...
for vi = 1 to (n+R1-1)/R1 do
  for si = 1 to R1 do
    for vj= 1 to (n+1+R2-1)/R2 do
      for sj= 1 to R2 do
        xCom[vj,sj,vi,si] =
          a*xCom[vj-1,sj-1,vi,si] + y[vi,si];
      end;
```

Общая методика преобразования

4 этап (Прокруст): расчет и подбор рациональных параметров реализации задачи

1. Первым критическим ресурсом при расчете параметров является скважность решения – необходимо обеспечить скважность, равную 1. Для этого в цепь обратной связи добавляются регистры для организации вычислений по схеме «конвейер конвейеров».
2. Следующий критический параметр - число каналов КРП. Масштабируем число необходимых для вычислений каналов до имеющегося количества добавлением регистров в цепь обратной связи.
3. Если имеющихся КРП недостаточно для реализации базового подграфа задачи, необходимо редуцировать производительность задачи по критическому ресурсу (процессор «Щелкунчик»).
4. После определения минимального числа каналов КРП для реализации задачи, рассчитывается число устройств для распаралелливания по итерациям (если это возможно по задаче), либо процедурно (перезапуском кадров).
5. Адаптация программы под рассчитанные параметры: обработка аргументов с итерационным циклом (через регистр или внутреннюю память), организация обратной связи, организация «конвейера в конвейере»