

CLAIMS

1. A method, comprising:
- loading a source binary application comprising a plurality of source binary code blocks for execution by a computer system, the computer system comprising memory and
- 5 a processing unit comprising N_t target registers, wherein the source binary application is configured to be executed in an environment having N_s source registers wherein N_t is less than N_s ;
- identifying, during runtime of said source binary application, a frequently executed source binary code block within said source binary application;
- 10 defining a region comprising said frequently executed source binary code block and at least one neighboring source binary code block that exceeds an affinity threshold, the frequently executed source binary code block and the at least one neighboring source binary code block both include instructions to access at least one source register;
- determining a frequency of access for each source register accessed in the region;
- 15 and
- mapping each of N most frequently accessed source registers in the region to a respective one of the N_t target registers wherein N is less than or equal to N_t .
2. The method of claim 1 further comprising generating an optimized target binary
- 20 code block corresponding to each source binary code block in the region based at least in part on said register mapping wherein each optimized target binary code block comprises a prologue configured to load data associated with each mapped source register from memory to a corresponding target register, a body comprising translated source binary code corresponding to the source binary code block and a jump on condition and at least
- 25 one epilogue, the jump on condition and the at least one epilogue configured to control

flow from a first optimized target binary code block in the region to a second target binary code block based, at least in part, on whether the second target binary code block is in the region.

- 5 3. The method of claim 2, wherein the plurality epilogues comprises a first epilogue and a second epilogue, the first epilogue is configured to be executed if the second target binary code block is a second optimized target binary code block in the region, the first epilogue comprising a first instruction to jump to a body of the second optimized target binary code block and the second epilogue is configured to be executed if the second
10 target binary code block is not in the region, the second epilogue comprising a second instruction to store each target register corresponding to a respective mapped source register to memory and a third instruction to jump to a beginning of the second target binary code block.
- 15 4. The method of claim 1, further comprising identifying live-in and live-out registers for the region comprising identifying live-in and live-out registers for each source binary code block in the region.
- 20 5. The method of claim 4, further comprising generating an optimized target binary code block corresponding to each source binary code block in the region based at least in part on said register mapping wherein each optimized target binary code block comprises a prologue configured to load data associated with each mapped and live-in source register from memory to a corresponding target register, a body comprising translated source binary code corresponding to the source binary code block and a jump on condition and at
25 least one epilogue, the jump on condition and the at least one epilogue configured to

control flow from a first optimized target binary code block in the region to a second target binary code block based, at least in part, on whether the second target binary code block is in the region.

5 6. The method of claim 2, further comprising:

executing a prologue of a first optimized target binary code block in the region,
said executing comprising loading each of N target registers with data corresponding to a
respective one of the N mapped source registers for the region;

10 executing a body of the first optimized target binary code block, the first optimized
target binary code block including a first branch instruction configured to jump to a
second target binary code block;

determining whether the second target binary code block is in the region; and
executing a first epilogue to jump to a body of a second optimized target code
block corresponding to the second target binary code block if the second target code block
15 is in the region or executing a second epilogue to jump to a beginning of the second target
code block if the second target code block is not in the region.

7. The method of claim 5, further comprising:

20 executing a prologue of a first optimized target binary code block in the region,
said executing comprising, for each mapped live-in source register in the region, loading
one target register with data from memory corresponding to the mapped live-in source
register;

executing a body of the first optimized target binary code block, the first optimized
target binary code block including a first branch instruction configured to jump to a
25 second target binary code block;

determining whether the second target binary code block is in the region; and
executing a first epilogue to jump to a body of the second target code block if the
second target code block is in the region or executing a second epilogue to store the
mapped live-out registers for the region in memory and jump to a beginning of the second
5 target code block if the second target code block is not in said region.

8. The method of claim 1, wherein affinity corresponds to a correlation in likelihood
of execution of the frequently executed source binary code block and the at least one
neighboring source binary code block.

10

9. The method of claim 1, wherein the frequently executed source binary code block
and the at least one neighboring source binary code block are each bounded by a branch
instruction.

15 10. An article, comprising a tangible storage medium having instructions stored
thereon which when executed by a processor result in the following operations
comprising:

loading a source binary application comprising a plurality of source binary code
blocks for execution, wherein the processor comprises N_t target registers, wherein the
20 source binary application is configured to be executed in an environment having N_s source
registers wherein N_t is less than N_s ;

identifying, during runtime of said source binary application, a frequently executed
source binary code block within said source binary application;

defining a region comprising said frequently executed source binary code block
25 and at least one neighboring source binary code block that exceeds an affinity threshold

wherein affinity corresponds to a correlation in likelihood of execution of the frequently executed source binary code block and the at least one neighboring source binary code block, the frequently executed source binary code block and the at least one neighboring source binary code block both include instructions to access at least one source register

5 and are each bounded by a branch instruction;

determining a frequency of access for each source register accessed in the region;

and

mapping each of N most frequently accessed source registers in the region to a respective one of the N_t target registers wherein N is less than or equal to N_t .

10

11. The article of claim 10, wherein the instructions that when executed by the processor result in the following additional operations comprising:

generating an optimized target binary code block corresponding to each source binary code block in the region based at least in part on said register mapping wherein

15 each optimized target binary code block comprises a prologue configured to load data associated with each mapped source register from memory to a corresponding target register, a body comprising translated source binary code corresponding to the source binary code block and a jump on condition and at least one epilogue, the jump on condition and the at least one epilogue configured to control flow from a first optimized
20 target binary code block in the region to a second target binary code block based, at least in part, on whether the second target binary code block is in the region.

12. The article of claim 11, wherein the plurality epilogues comprises a first epilogue and a second epilogue, the first epilogue is configured to be executed if the second target

25 binary code block is a second optimized target binary code block in the region, the first

epilogue comprising a first instruction to jump to a body of the second optimized target binary code block and the second epilogue is configured to be executed if the second target binary code block is not in the region, the second epilogue comprising a second instruction to store each target register corresponding to a respective mapped source register to memory and a third instruction to jump to a beginning of the second target binary code block.

13. The article of claim 10, wherein the instructions that when executed by the processor result in the following additional operations comprising:

10 identifying live-in and live-out registers for the region comprising identifying live-in and live-out registers for each source binary code block in the region.

14. The article of claim 13, wherein the instructions that when executed by the processor result in the following additional operations comprising:

15 generating an optimized target binary code block corresponding to each source binary code block in the region based at least in part on said register mapping wherein each optimized target binary code block comprises a prologue configured to load data associated with each mapped and live-in source register from memory to a corresponding target register, a body comprising translated source binary code corresponding to the source binary code block and a jump on condition and at least one epilogue, the jump on condition and the at least one epilogue configured to control flow from a first optimized target binary code block in the region to a second target binary code block based, at least in part, on whether the second target binary code block is in the region.

20

25 15. The article of claim 11, wherein the instructions that when executed by the

processor result in the following additional operations comprising:

executing a prologue of a first optimized target binary code block in the region,
said executing comprising loading each of N target registers with data corresponding to a
respective one of the N mapped source registers for the region;

5 executing a body of the first optimized target binary code block, the first optimized
target binary code block including a first branch instruction configured to jump to a
second target binary code block;

 determining whether the second target binary code block is in the region; and

 executing a first epilogue to jump to a body of a second optimized target code
10 block corresponding to the second target binary code block if the second target code block
is in the region or executing a second epilogue to jump to a beginning of the second target
code block if the second target code block is not in the region.

16. The article of claim 14, wherein the instructions that when executed by the
15 processor result in the following additional operations comprising:

 executing a prologue of a first optimized target binary code block in the region,
said executing comprising, for each mapped live-in source register in the region, loading
one target register with data from memory corresponding to the mapped live-in source
register;

20 executing a body of the first optimized target binary code block, the first optimized
target binary code block including a first branch instruction configured to jump to a
second target binary code block;

 determining whether the second target binary code block is in the region; and

 executing a first epilogue to jump to a body of the second target code block if the
25 second target code block is in the region or executing a second epilogue to store the

mapped live-out registers for the region in memory and jump to a beginning of the second target code block if the second target code block is not in said region.

17. A system, comprising:

5 a processor comprising N_t target registers; and

a memory configured to store a dynamic binary translator, a register mapping module and a source binary application configured to be executed in an environment having N_s source registers wherein N_t is less than N_s , the source binary application comprising a plurality of source binary code blocks to be executed on the processor,

10 wherein when the dynamic binary translator and register mapping module are executed on the processor at least one of the dynamic binary translator and register mapping module cause the processor to:

identify, during runtime of said source binary application, a frequently executed source binary code block within said source binary application;

15 define a region comprising said frequently executed source binary code block and at least one neighboring source binary code block that exceeds an affinity threshold, the frequently executed source binary code block and the at least one neighboring source binary code block both include instructions to access at least one source register;

determine a frequency of access for each source register accessed in the region;

20 and

map each of N most frequently accessed source registers in the region to a respective one of the N_t target registers wherein N is less than or equal to N_t .

18. The system of claim 17, wherein the register mapping module further causes the
25 processor to generate an optimized target binary code block corresponding to each source

binary code block in the region based at least in part on said register mapping wherein each optimized target binary code block comprises a prologue configured to load data associated with each mapped source register from memory to a corresponding target register, a body comprising translated source binary code corresponding to the source

5 binary code block and a jump on condition and at least one epilogue, the jump on condition and the at least one epilogue configured to control flow from a first optimized target binary code block in the region to a second target binary code block based, at least in part, on whether the second target binary code block is in the region.

10 19. The system of claim 18, wherein the plurality epilogues comprises a first epilogue and a second epilogue, the first epilogue is configured to be executed if the second target binary code block is a second optimized target binary code block in the region, the first epilogue comprising a first instruction to jump to a body of the second optimized target binary code block and the second epilogue is configured to be executed if the second
15 target binary code block is not in the region, the second epilogue comprising a second instruction to store each target register corresponding to a respective mapped source register to memory and a third instruction to jump to a beginning of the second target binary code block.

20 20. The system of claim 17, wherein the register mapping module further causes the processor to identify live-in and live-out registers for the region comprising identifying live-in and live-out registers for each source binary code block in the region.

21. The article of claim 10, wherein affinity corresponds to a correlation in likelihood
25 of execution of the frequently executed source binary code block and the at least one

neighboring source binary code block.

22. The article of claim 10, wherein the frequently executed source binary code block
and the at least one neighboring source binary code block are each bounded by a branch
5 instruction.

23. The system of claim 17, wherein affinity corresponds to a correlation in likelihood
of execution of the frequently executed source binary code block and the at least one
neighboring source binary code block.
10

24. The system of claim 17, wherein the frequently executed source binary code block
and the at least one neighboring source binary code block are each bounded by a branch
instruction.

CLAIMS

1. A method, comprising:

loading a source binary application comprising a plurality of source binary code blocks for execution by a computer system, the computer system comprising memory and a processing unit comprising N_t target registers, wherein the source binary application is configured to be executed in an environment having N_s source registers wherein N_t is less than N_s ;

identifying, during runtime of said source binary application, a frequently executed source binary code block within said source binary application;

defining a region comprising said frequently executed source binary code block and at least one neighboring source binary code block that exceeds an affinity threshold, the frequently executed source binary code block and the at least one neighboring source binary code block both include instructions to access at least one source register in common, and the frequently executed source binary code block and the at least one neighboring source binary code block are each bounded by a branch instruction;

determining a frequency of access for each source register accessed in the region; and mapping each of N (N is greater than zero) most frequently accessed source registers in the region to a respective one of the N_t target registers wherein N is less than or equal to N_t .

2. The method of claim 1 further comprising generating an optimized target binary code block corresponding to each source binary code block in the region based at least in part on said register mapping wherein each optimized target binary code block comprises a prologue configured to load data associated with each mapped source register from memory to a corresponding target register, a body comprising translated source binary code corresponding to the source binary code block and a jump on condition and at least one epilogue, the jump on condition and the at least one epilogue configured to control flow from a first optimized target binary code block in the region to a second target binary code block based, at least in part, on whether the second target binary code block is in the region.

3. The method of claim 2, wherein the at least one epilogue comprises a first epilogue and a

second epilogue, the first epilogue is configured to be executed if the second target binary code block is a second optimized target binary code block in the region, the first epilogue comprising a first instruction to jump to a body of the second optimized target binary code block and the second epilogue is configured to be executed if the second target binary code block is not in the region, the second epilogue comprising a second instruction to store each target register corresponding to a respective mapped source register to memory and a third instruction to jump to a beginning of the second target binary code block.

4. The method of claim 1, further comprising identifying live-in and live-out registers for the region comprising identifying live-in and live-out registers for each source binary code block in the region.

5. The method of claim 4, further comprising generating an optimized target binary code block corresponding to each source binary code block in the region based at least in part on said register mapping wherein each optimized target binary code block comprises a prologue configured to load data associated with each mapped and live-in source register from memory to a corresponding target register, a body comprising translated source binary code corresponding to the source binary code block and a jump on condition and at least one epilogue, the jump on condition and the at least one epilogue configured to control flow from a first optimized target binary code block in the region to a second target binary code block based, at least in part, on whether the second target binary code block is in the region.

6. The method of claim 2, further comprising:

executing a prologue of a first optimized target binary code block in the region, said executing comprising loading each of N target registers with data corresponding to a respective one of the N mapped source registers for the region;

executing a body of the first optimized target binary code block, the first optimized target binary code block including a first branch instruction configured to jump to a second target binary code block;

determining whether the second target binary code block is in the region; and

executing a first epilogue to jump to a body of a second optimized target code block corresponding to the second target binary code block if the second target code block is in the region or executing a second epilogue to jump to a beginning of the second target code block if the second target code block is not in the region.

7. The method of claim 5, further comprising:

executing a prologue of a first optimized target binary code block in the region, said executing comprising, for each mapped live-in source register in the region, loading one target register with data from memory corresponding to the mapped live-in source register;

executing a body of the first optimized target binary code block, the first optimized target binary code block including a first branch instruction configured to jump to a second target binary code block;

determining whether the second target binary code block is in the region; and

executing a first epilogue to jump to a body of the second target code block if the second target code block is in the region or executing a second epilogue to store the mapped live-out registers for the region in memory and jump to a beginning of the second target code block if the second target code block is not in said region.

8. The method of claim 1, wherein affinity corresponds to a correlation in likelihood of execution of the frequently executed source binary code block and the at least one neighboring source binary code block.

9. An article, comprising a non-transitory tangible storage medium having instructions stored thereon which when executed by a processor result in the following operations comprising:

loading a source binary application comprising a plurality of source binary code blocks for execution, wherein the processor comprises N_t target registers, wherein the source binary application is configured to be executed in an environment having N_s source registers wherein N_t is less than N_s ;

identifying, during runtime of said source binary application, a frequently executed source binary code block within said source binary application;

defining a region comprising said frequently executed source binary code block and at least one neighboring source binary code block that exceeds an affinity threshold wherein affinity corresponds to a correlation in likelihood of execution of the frequently executed source binary code block and the at least one neighboring source binary code block, the frequently executed source binary code block and the at least one neighboring source binary code block both include instructions to access at least one source register in common and the frequently executed source binary code block and the at least one neighboring source binary code block are each bounded by a branch instruction;

determining a frequency of access for each source register accessed in the region; and

mapping each of N (N is greater than zero) most frequently accessed source registers in the region to a respective one of the N_t target registers wherein N is less than or equal to N_t .

10. The article of claim 9, wherein the instructions that when executed by the processor result in the following additional operations comprising:

generating an optimized target binary code block corresponding to each source binary code block in the region based at least in part on said register mapping wherein each optimized target binary code block comprises a prologue configured to load data associated with each mapped source register from memory to a corresponding target register, a body comprising translated source binary code corresponding to the source binary code block and a jump on condition and at least one epilogue, the jump on condition and the at least one epilogue configured to control flow from a first optimized target binary code block in the region to a second target binary code block based, at least in part, on whether the second target binary code block is in the region.

11. The article of claim 10, wherein the at least one epilogue comprises a first epilogue and a second epilogue, the first epilogue is configured to be executed if the second target binary code block is a second optimized target binary code block in the region, the first epilogue comprising a first instruction to jump to a body of the second optimized target binary code block and the second epilogue is configured to be executed if the second target binary code block is not in the region, the second epilogue comprising a second instruction to store each target register corresponding to a respective mapped source register to memory and a third

instruction to jump to a beginning of the second target binary code block.

12. The article of claim 9, wherein the instructions that when executed by the processor result in the following additional operations comprising:

identifying live-in and live-out registers for the region comprising identifying live-in and live-out registers for each source binary code block in the region.

13. The article of claim 12, wherein the instructions that when executed by the processor result in the following additional operations comprising:

generating an optimized target binary code block corresponding to each source binary code block in the region based at least in part on said register mapping wherein each optimized target binary code block comprises a prologue configured to load data associated with each mapped and live-in source register from memory to a corresponding target register, a body comprising translated source binary code corresponding to the source binary code block and a jump on condition and at least one epilogue, the jump on condition and the at least one epilogue configured to control flow from a first optimized target binary code block in the region to a second target binary code block based, at least in part, on whether the second target binary code block is in the region.

14. The article of claim 10, wherein the instructions that when executed by the processor result in the following additional operations comprising:

executing a prologue of a first optimized target binary code block in the region, said executing comprising loading each of N target registers with data corresponding to a respective one of the N mapped source registers for the region;

executing a body of the first optimized target binary code block, the first optimized target binary code block including a first branch instruction configured to jump to a second target binary code block;

determining whether the second target binary code block is in the region; and

executing a first epilogue to jump to a body of a second optimized target code block corresponding to the second target binary code block if the second target code block is in the

region or executing a second epilogue to jump to a beginning of the second target code block if the second target code block is not in the region.

15. The article of claim 13, wherein the instructions that when executed by the processor result in the following additional operations comprising:

executing a prologue of a first optimized target binary code block in the region, said executing comprising, for each mapped live-in source register in the region, loading one target register with data from memory corresponding to the mapped live-in source register;

executing a body of the first optimized target binary code block, the first optimized target binary code block including a first branch instruction configured to jump to a second target binary code block;

determining whether the second target binary code block is in the region; and

executing a first epilogue to jump to a body of the second target code block if the second target code block is in the region or executing a second epilogue to store the mapped live-out registers for the region in memory and jump to a beginning of the second target code block if the second target code block is not in said region.

16. A system, comprising:

a processor comprising N_t target registers; and

a memory configured to store a dynamic binary translator, a register mapping module and a source binary application configured to be executed in an environment having N_s source registers wherein N_t is less than N_s , the source binary application comprising a plurality of source binary code blocks to be executed on the processor, wherein when the dynamic binary translator and register mapping module are executed on the processor at least one of the dynamic binary translator and register mapping module cause the processor to:

identify, during runtime of said source binary application, a frequently executed source binary code block within said source binary application;

define a region comprising said frequently executed source binary code block and at least one neighboring source binary code block that exceeds an affinity threshold, the

frequently executed source binary code block and the at least one neighboring source binary code block both include instructions to access at least one source register in common, and the frequently executed source binary code block and the at least one neighboring source binary code block are each bounded by a branch instruction;

determine a frequency of access for each source register accessed in the region; and

map each of N (N is greater than zero) most frequently accessed source registers in the region to a respective one of the N_t target registers wherein N is less than or equal to N_t .

17. The system of claim 16, wherein the register mapping module further causes the processor to generate an optimized target binary code block corresponding to each source binary code block in the region based at least in part on said register mapping wherein each optimized target binary code block comprises a prologue configured to load data associated with each mapped source register from memory to a corresponding target register, a body comprising translated source binary code corresponding to the source binary code block and a jump on condition and at least one epilogue, the jump on condition and the at least one epilogue configured to control flow from a first optimized target binary code block in the region to a second target binary code block based, at least in part, on whether the second target binary code block is in the region.

18. The system of claim 17, wherein the at least one epilogue comprises a first epilogue and a second epilogue, the first epilogue is configured to be executed if the second target binary code block is a second optimized target binary code block in the region, the first epilogue comprising a first instruction to jump to a body of the second optimized target binary code block and the second epilogue is configured to be executed if the second target binary code block is not in the region, the second epilogue comprising a second instruction to store each target register corresponding to a respective mapped source register to memory and a third instruction to jump to a beginning of the second target binary code block.

19. The system of claim 16, wherein the register mapping module further causes the processor to identify live-in and live-out registers for the region comprising identifying live-in and live-out registers for each source binary code block in the region.

20. The article of claim 9, wherein affinity corresponds to a correlation in likelihood of execution of the frequently executed source binary code block and the at least one neighboring source binary code block.

21. The system of claim 16, wherein affinity corresponds to a correlation in likelihood of execution of the frequently executed source binary code block and the at least one neighboring source binary code block.